TATA CONSULTANCY SERVICES
Experience certainty.

Where ABAP meets HANA

- Swetha Shivakumar

Contents

- ABAP Meets SAP HANA: Evolution and Architecture
- Where ABAP meets HANA
- Open SQL
- CDS – Core Data Services
- AMDP's – ABAP Managed Database Procedures

TATA CONSULTANCY SERVICES
Experience certainty.

ABAP Meets SAP HANA: Evolution and Architecture

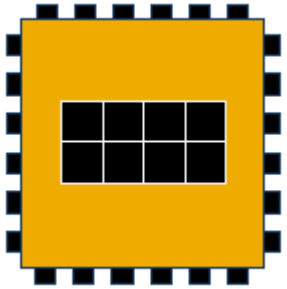
ABAP Meets SAP HANA: Evolution and Architecture

- Hardware & Software Innovations
- Evolution of ABAP and SAP HANA
- Introduction to ABAP for SAP HANA Architecture
- Highlights of SAP NetWeaver Application Server ABAP 7.4

Hardware Technology Innovations

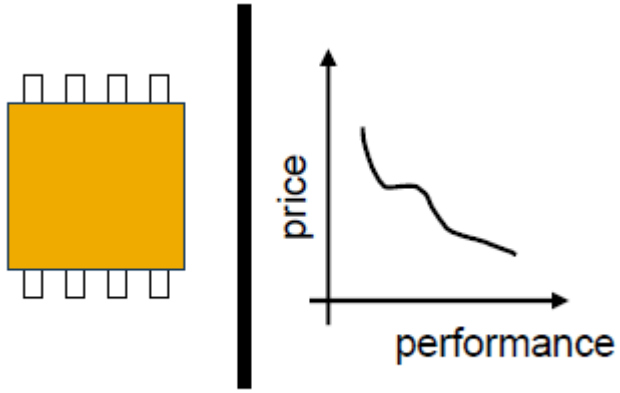
Multicore Architecture

- 8 CPUs x (8-16) cores per blade
- Parallel scaling with many blades



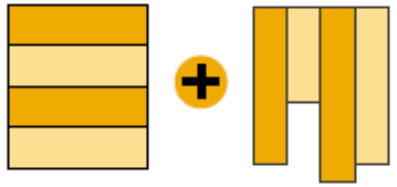
Address Space

- 64-bit address space (4TB in current server boards)
- Dramatic decline in price



SAP Software Technology Innovations

Row and Column Store



Data Compression

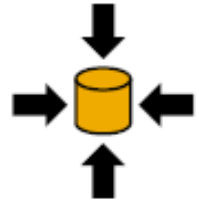
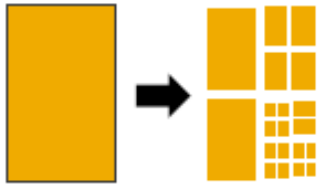
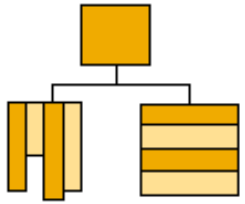


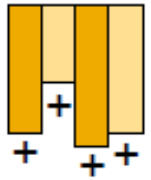
Table Partitioning



Avoidance of Aggregate Tables



Insert Only on Delta



Introduction to ABAP for SAP HANA Architecture

Few features contained in AS ABAP 7.4:

- SAPUI5 and SAP Gateway capabilities
- Easy mobile access and mobile applications development
- More Declarative and Functional Coverage
- Broader coverage of the SQL standard by Open SQL
- Advanced view building with ABAP Core Data Services
- ABAP-managed database procedures
- Real-time eventing using the ABAP Channels
- End-to-end developer experience in Eclipse

- **The 3 essential ingredients in the context of ABAP developments for SAP HANA are**
  - ✓ the **SAP HANA** platform
  - ✓ the **"Code-to-Data"** programming paradigm in ABAP
  - ✓ and the **Eclipse** development platform.

SAP HANA Platform and Eclipse

SAP HANA platform

- RDBMS
- In-memory Computing
- Columnar and row-based store
- Multi Core CPU's
- Convergence of OLTP and OLAP
- Specialized engines and application libraries like Predictive Analysis, Text mining and Search

Eclipse:

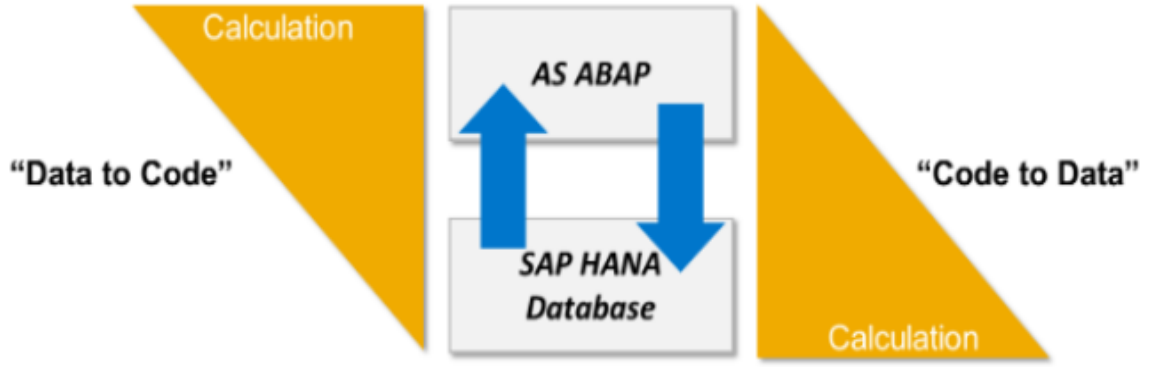
- Integrated development Environment(IDE)
- Uniform Developer experience
- Available SAP Development Tools
  - SAP HANA, Cloud, BW, Gateway and ABAP with the ABAP Development Tools for SAP NetWeaver (aka ABAP in Eclipse)

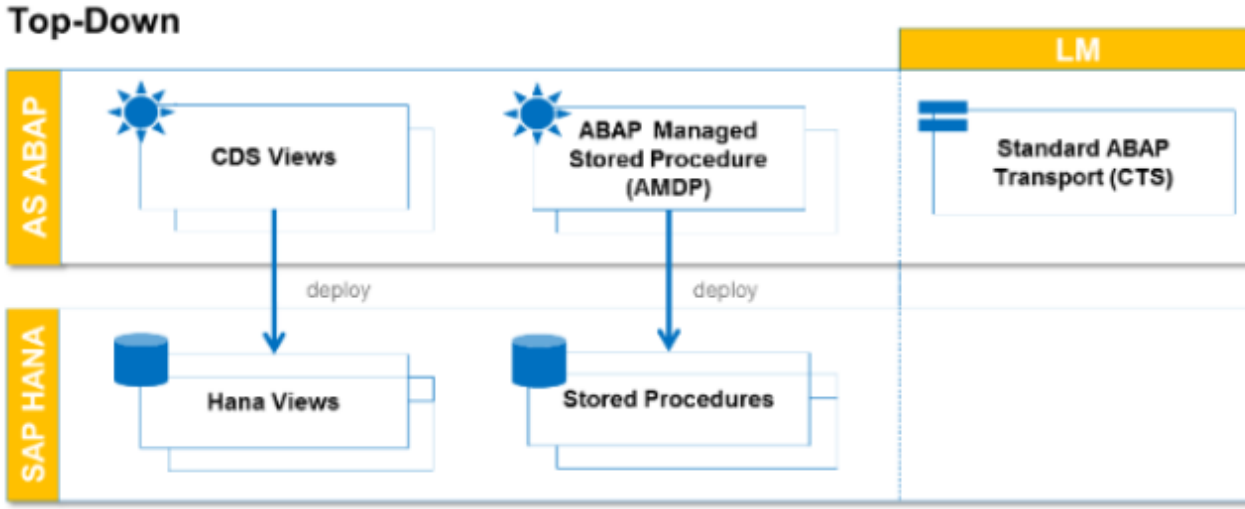
Note:

The ABAP development tools for eclipse are **mandatory** when it comes to full support of ABAP developments for SAP HANA

Code-To-Data Paradigm

- Before: Data-To-Code
- After: Code-to-Data
- What is Code Push Down?
- Where does Code Push Down Start?
- Bottom-Up-Approach
- Top-Down-Approach





Recommended approach

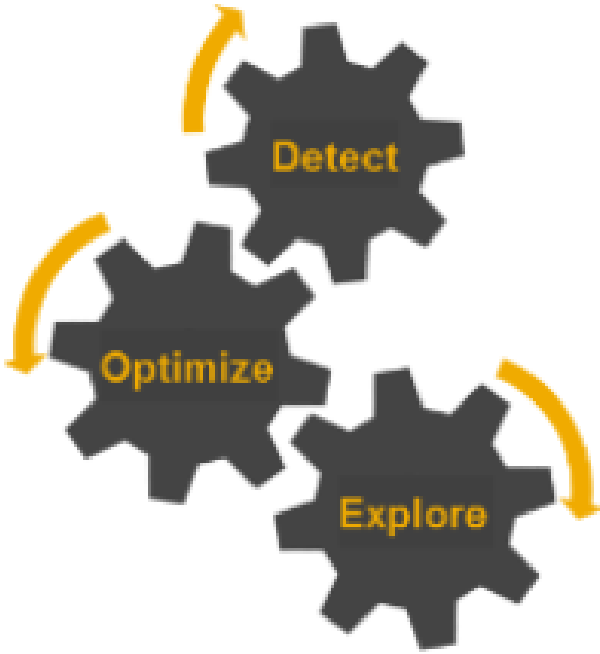
How custom ABAP code benefits from SAP HANA?

ABAP development guidelines

- Improved tools for performance analysis
- Extensions to ABAP and OpenSQL
- Features for SAP HANA-oriented programming
- Re-use components optimized for SAP HANA

Transparent optimizations

- Fast Data Access (new data exchange protocol)
- optimized SELECT... INTO ITAB and SELECT SINGLE
- optimized FOR ALL ENTRIES-clause



- The typical and recommended approach for customers who want to migrate their custom solutions to SAP HANA (or have already done so) follows the pattern "Detect - Optimize - Explore".
  - **Detect** your custom ABAP code to be adapted making use of the advanced quality assurance tooling
  - **Optimize** your custom ABAP code for SAP HANA by making use of the various programming features
  - **Explore** the new opportunities provided by SAP HANA and integrate them easily in your custom ABAP applications

Detect ABAP custom code to be adapted

"Will my custom ABAP code still work on SAP HANA?"

The question mainly relates to two areas in which code corrections and adaptations may be required before the migration in order to avoid regressions:

- Functional correctness
- (SQL) Performance.

- **What should be detected?**
- **Which support do I get from SAP?**

To assist you in your different detection activities, SAP NetWeaver AS ABAP 7.4 provides advanced quality assurance tooling. Three major tools can be named:

- ABAP Test Cockpit (transaction SATC)
- SQL Monitor (transaction SQLM)
- SQL Performance Tuning Worklist (transaction SWLT)

Quality Assurance Tooling

ABAP Test Cockpit

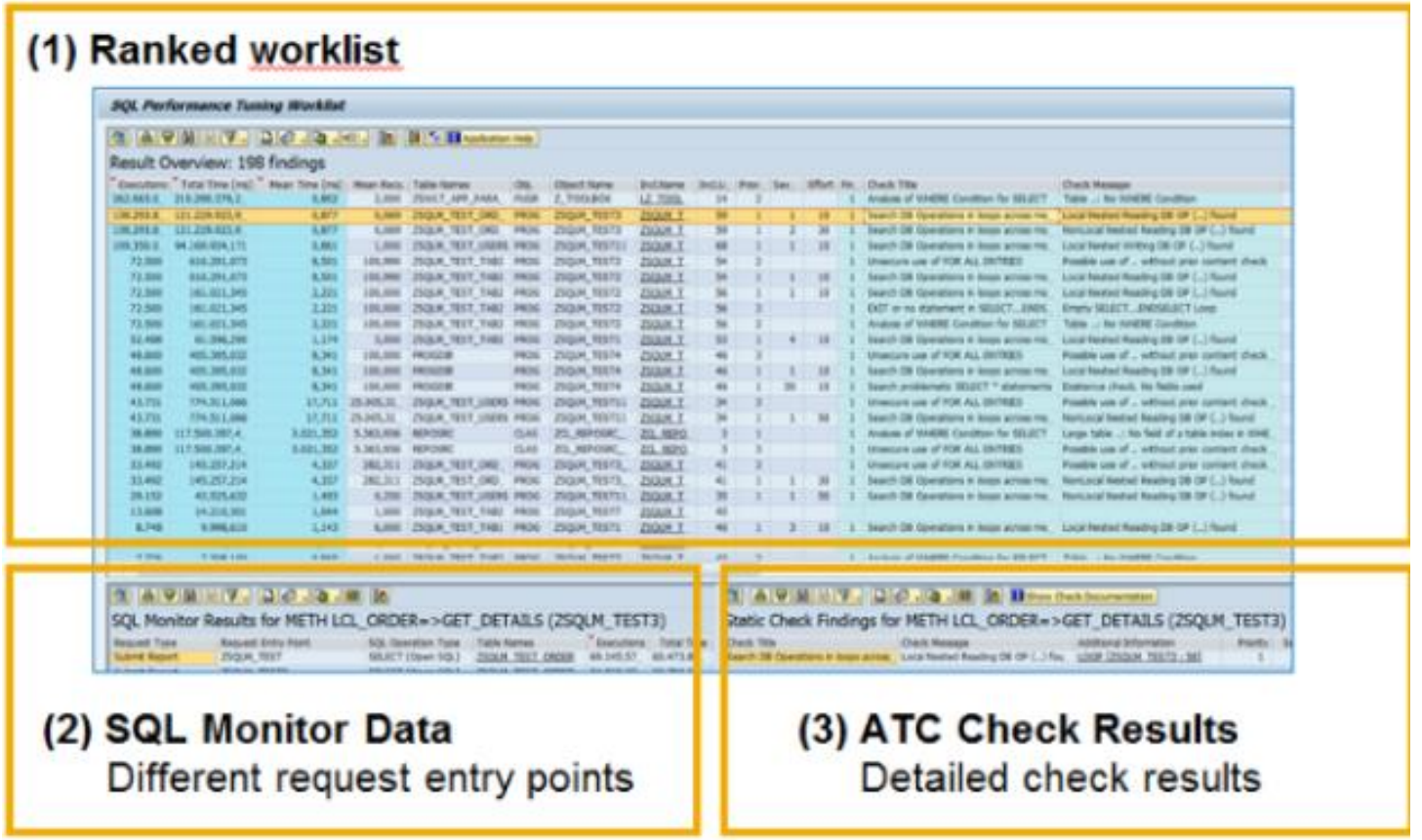
- The ABAP Test Cockpit (ATC) is the standard tool for running static code checks on ABAP development objects

SQL Monitor

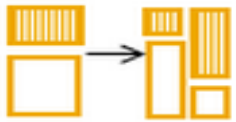
- SQL Monitor is the recommended standard performance analysis tool when it comes to the runtime monitoring of productive systems without impacting the running business processes.

Performance Worklist Tool

- The SQL Performance Worklist (Transaction: SWLT) allows you to correlate SQL runtime data with ABAP code analysis to plan optimizations using results from the new SQLMonitor and CodeInspector.



The Golden Rules

Icon	Rule	Details / Examples
	**Keep the result sets small**	• Do not retrieve rows from the database and discard them on the application server using CHECK or EXIT, e.g. in SELECT loops • Make the WHERE clause as specific as possible
	**Minimize amount of transferred data**	• Use SELECT with a field list instead of SELECT * in order to transfer just the columns you really need • Use aggregate functions (COUNT, MIN, MAX, SUM, AVG) instead of transferring all the rows to the application server
	**Minimize the number of data transfers**	• Use JOINs and / or sub-queries instead of nested SELECT loops • Use SELECT … FOR ALL ENTRIES instead of lots of SELECTs or SELECT SINGLEs • Use array variants of INSERT, UPDATE, MODIFY, and DELETE
	**Minimize the search overhead**	• Define and use appropriate secondary indexes
	**Keep load away from the database**	• Avoid reading data redundantly • Use table buffering (if possible) and do not bypass it • Sort Data in Your ABAP Programs

Additions in the context of SAP HANA

Rule 1 : Keep the result set small

Guideline 	Keep the result sets small
Additions in the context of SAP HANA	Don't retrieve rows from the database and discard them on the application server using CHECK or EXIT, e.g. SELECT loops • Make a SELECT statements as specific as possible • If possible, replace this kind of statement with a JOIN

Rule 2 : Minimize amount of transferred data

Guideline 	Minimize amount of transferred data
Additions in the context of SAP HANA	Use SELECT with field list instead of SELECT * in order to transfer just the column you really need • Use aggregation functions (COUNT, MIN, MAX, SUM, AVG) instead of transferring all the rows to the application server • Replace sort and group operations in the application logic with standard SQL statements, e.g. with GROUP and ORDER BY • To calculate groups including subtotals on SAP HANA use e.g. GROUP BY GROUPING SETS WITH SUBTOTAL STRUCTURED RESULT WITH OVERVIEW PREFIX

Rule 3 : Avoide many small request and selecting not needed columns

Guideline 	Minimize the number of data transfers
Additions in the context of SAP HANA	On all database systems, there is a small performance overhead associated with every request for connection handling, SQL parsing, execution plan determination, etc. In particular, the following existing guidelines should be priorized higher on SAP HANA: • For modifying operations (INSERT, UPDATE, DELETE) using array opreations should be preferred to single operations when changing many data records. • Nested SELECT loops should be avoided or replaced if possible by o Changing nested SELECT statement to an appropriate SQL construct (e.g. SELECT...FOR ALL ENTRIES, JOIN, sub-query) o Using the ABAP table buffer (see existing guidelines for the ABAP table buffer) In addition, reducing the selected field list to the actual need colums should be priorized higher an SAP HANA in case of statements which are either executed very frequently, or which return large result sets.

Additions in the context of SAP HANA

Rule 4 : Define and use appropriate secondary indexes

Guideline	Minimize the Search Overhead
Additions in the context of SAP HANA	In most cases SAP HANA does not require scondary indices for good search performance. To reduce main memory consumption and to improve insert performance all existing non-unique secondary database indices on columnar tables are improved during migration or are not created during instalation for all AS ABAP systems from SAP NetWeaver 7.4 onwards. Some use cases can benefit from secondary indexes. This is mainly true for highly selective queries on non-primary key fields. Tese queries can be improved significant by indexes on most selective fields (see SAP Note 1794297 for more details).

Rule 5 : Keep unnecessary load away from the database

Guideline	Let the databsae do the job
Additions in the context of SAP HANA	On SAP HANA, it is beneficial to move data-intensive calculations to the database. Nevertheless, it is not recommended to execute the same operation redundantly, e.g. in different user contexts or different dialog steps of the user. Meaningful buffering of results on the application server should be applied. The following existing recommendations should be considered in this light on SAP HANA • Sorting data: In general, the recommendations for sorting remain as before, i.e. if the database does not use the same index for sorting as for selection, then it may be more efficient to sort data in the application server. However, of on SAP HANA the sorting is part of a larger calculation logic, (e.g. within a procedure), it should be done on SAP HANA. • Logical database: The general guidelines regarding logical databases remain the same. However, using logical databases does not provide performance advantages compared to a proper usage of Open SQL.

Where ABAP meets HANA

- SAP has brought ABAP and SAP HANA together with features that enable developers to leverage the best of both worlds
- **New features**
  - ✓ **Open SQL** : Database-independent programming
  - ✓ **CDS** - core data services : unified data modelling and definition with
  - ✓ **AMDP** - ABAP-managed database procedures : simplified control of stored procedures

TATA CONSULTANCY SERVICES
Experience certainty.

Where ABAP meets HANA

- In Elaboration

Open SQL

Content

- Reminder: What is Open SQL?
- Code-to-Data using OPEN SQL
- What's New in Open SQL?
- New OPEN SQL syntax and its Enhancements
- Open SQL enhancements
  - Syntax
  - SELECT list
  - GROUP BY and HAVING clauses
  - JOIN statements
  - Client handling

Open SQL

Main advantages :

- its database-platform-independence;
- its integration into the ABAP language
- and runtime environment through internal tables;
- and strong support of SAP NetWeaver Application Server (SAP NetWeaver AS) ABAP with table buffering, SQL statement caching, and cursor caching.

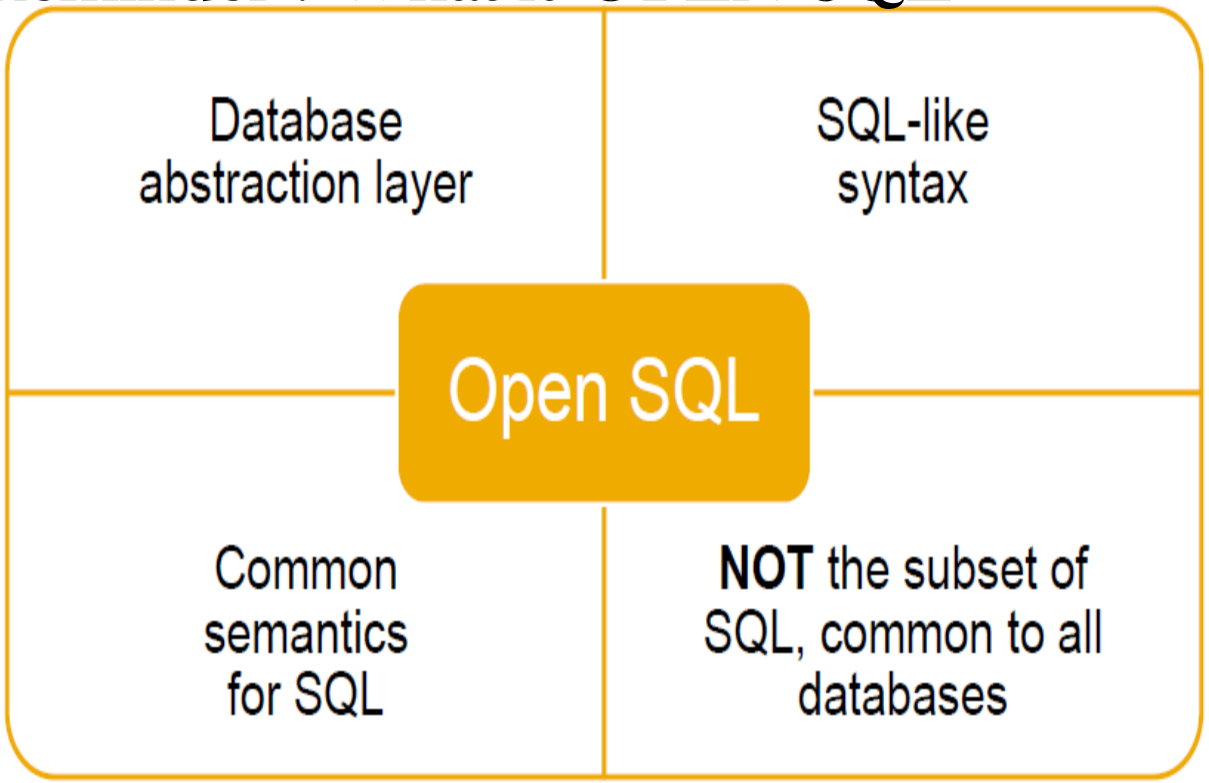
Open SQL aspires to

- enable the application of the Code-to-Data paradigm
- provide (more) standard SQL features
- enable the consumption of SAP HANA-specific features

Introduction of new features

Reduction of existing limitations

Reminder : What is OPEN SQL



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

Code-To-Data using OPEN SQL

5 **Ways of achieving Code Push Down to HANA from ABAP using Open SQL:**

- Start using the **new Open SQL syntax**. Remember it is easy to convert your existing Open SQL statements to new syntax without any side effects.
- Use **aggregate functions** where relevant instead of doing the aggregations in the ABAP layer
- Use **arithmetic and string expressions** within Open SQL statements instead of looping over the data fetched to do the arithmetic and string operations
- Use **computed columns** in order to push down computations that would otherwise be done in a long loops
- Use **CASE and/or IF..ELSE expressions within the Open SQL** in order embed the logic of conditional expression which in the past would have been done after fetching the results from the database.

What's New in Open SQL?

New Open SQL Syntax

- Escaping of host variables
- Comma separated element list

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

Target Type Inference

New SELECT List Features

- Aggregation functions
- Literal values
- Arithmetic expressions
- String expression

```
SELECT bp_id,
       company_name,
       so~currency_code,
       SUM( so~gross_amount )
          AS total_gross_amount
  FROM snwd_so AS so
  INNER JOIN snwd_bpa AS bpa
          ON so~buyer_guid = bpa~node_key
  INTO TABLE @DATA(lt_result)
  GROUP BY bp_id, company_name,
           so~currency_code.
```

New OPEN SQL syntax and its Enhancements

- Enhancements for Open SQL
- New SQL expressions and inline declarations.
  - Literal Values & Generic Existence Check
  - Expressions
  - Conditional Expressions
  - Expressions in GROUP BY & HAVING clause
  - Arithmetic Expressions
  - Support for JOIN statements
  - Automatic client Handling
- It also includes CDS.

```
TYPES BEGIN OF demo_structure.
  INCLUDE TYPE snwd_so AS so RENAMING WITH SUFFIX _so.
  INCLUDE TYPE snwd_bpa AS bpa RENAMING WITH SUFFIX _bp.
  INCLUDE TYPE snwd_so_inv_head AS head RENAMING WITH SUFFIX
      _head.
TYPES END OF demo_structure.
DATA: lt_data TYPE STANDARD TABLE OF demo_structure.

 SELECT so~*, bp~*, soi_head~*
   FROM snwd_so AS so
   INNER JOIN snwd_bpa AS bp
        ON so~buyer_guid = bp~node_key
   LEFT OUTER JOIN snwd_so_inv_head AS soi_head
         ON so~node_key = soi_head~so_guid
   INTO TABLE @lt_data UP TO 10 ROWS.
```

Figure 1 — New Open SQL syntax in an example ABAP program

Literal Values & Generic Existence Check

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.
```

Expressions

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).
```

String Expressions

- Concatenate character columns with the && operator

```
SELECT company_name
    && ' (' && legal_form && ')'
  FROM snwd_bpa
  INTO TABLE @DATA(lt_result).
```

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).
```

COALESCE Expression

```
SELECT so_id,
       so~gross_amount
              AS so_amount,
       inv_head~gross_amount
              AS inv_amount,
       "potential invoice amount
       COALESCE( inv_head~gross_amount,
                 so~gross_amount )
              AS expected_amount
  FROM snwd_so AS so
  LEFT OUTER JOIN
       snwd_so_inv_head AS inv_head
    ON inv_head~so_guid = so~node_key
  INTO TABLE @DATA(lt_result).
```

Expressions in GROUP BY & HAVING Clauses

GROUP BY Clause

```
SELECT bp_id,
       company_name,
       so~currency_code,
       SUM( so~gross_amount )
         AS total_amount,
       CASE
         WHEN so~gross_amount < 1000
           THEN 'X'
         ELSE ' '
       END AS low_volume_flag,
       COUNT( * ) AS cnt_orders
 FROM snwd_so AS so
 INNER JOIN snwd_bpa AS bpa
 ON bpa~node_key = so~buyer_guid
 INTO TABLE @DATA(lt_result)
 GROUP BY
   bp_id, company_name,
   so~currency_code,
   CASE
     WHEN so~gross_amount < 1000
       THEN 'X'
       ELSE ' '
     END
  ORDER BY company_name.
```

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.
```

Support for JOIN Statements

Enhancements

- Now available: RIGHT OUTER JOIN
- Enhanced bracketing in JOIN expressions
- New functionality in ON conditions of JOIN statements like:
  - Necessary requirement of a field of the right table in the ON condition is dropped
  - Operators like BETWEEN or “>” can be used for comparisons
  - Possibility to use fields of the right table in the WHERE clause of LEFT OUTER JOINs
- Restriction of maximum number of tables in JOINs has been increased to 50

```
SELECT
    so_id,
    bp_id,
    gross_amount
 FROM snwd_so AS so
 RIGHT OUTER JOIN snwd_bpa AS bpa
   ON so~buyer_guid = bpa~node_key
  AND so~gross_amount > 100000
 INTO TABLE @DATA(lt_result).
```

Automatic Client Handling

Automatic Client Handling

- Well known Open SQL client handling
- Client handling can be overruled with **USING CLIENT**
- Simplified/improved client handling in JOINs

```
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).
```

CDS Views: Core Data Services

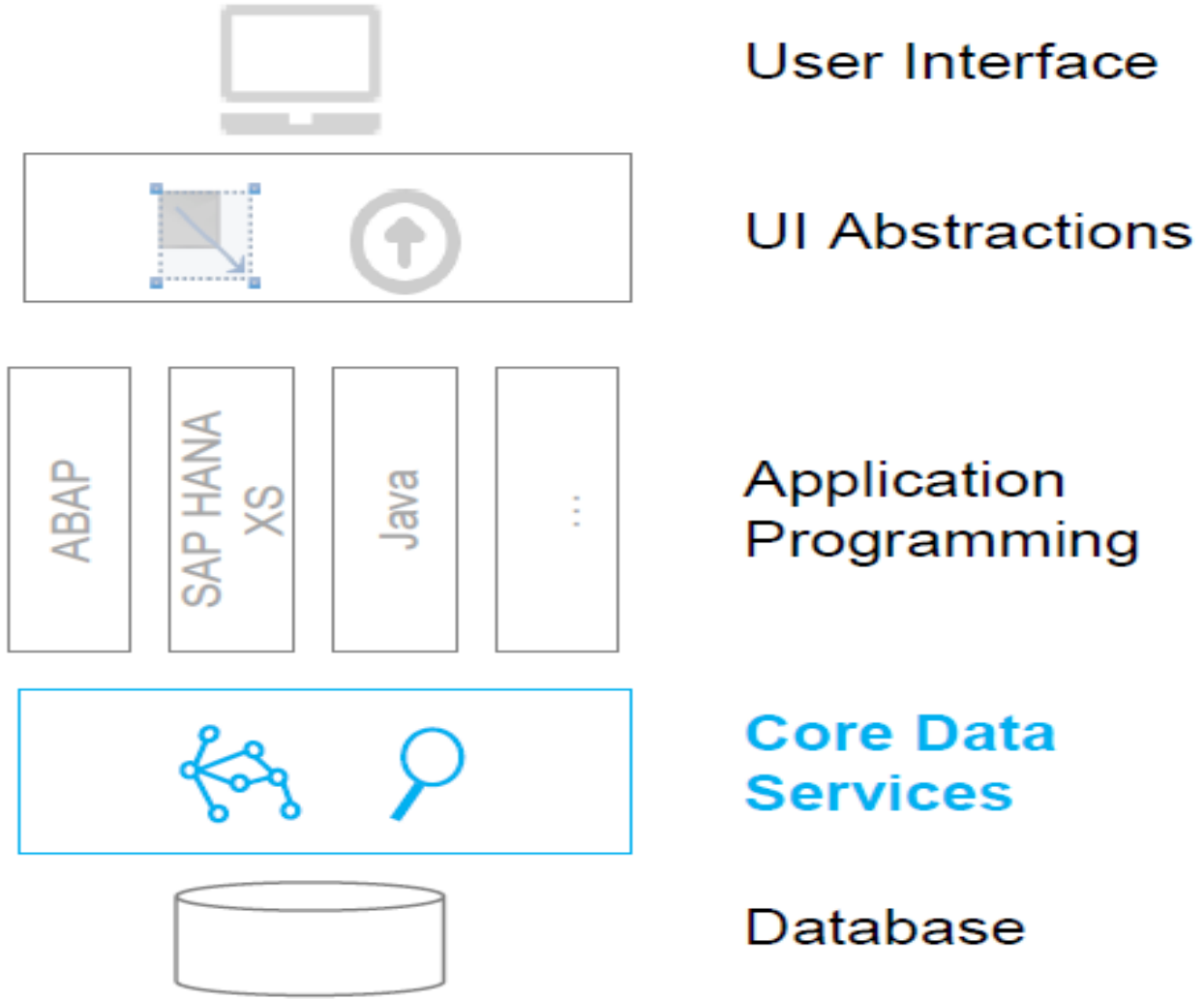
Content

- Introduction to Core Data Services (CDS)
- Core Data Services in ABAP
- Consumption of CDS views
- CDS vs. Open SQL
- Simple CDS View
- CDS View Definition Features
- Projection List
- Alias
- View-on-View Concept
- CDS View Extensions
- CDS Views with Input Parameters
- Additional Information

Core Data Services: What's in It for You?

Advantages

- Semantically rich data models.
- Compatibility across any database platform
- Efficiency
- Support for annotations
- Support for conceptual associations
- Extensibility



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**)

Code-to-Data Paradigm for CDS

The Code-to-Data paradigm is supported through e.g. various built-in functions, unions, associations and path expressions. ABAP CDS is integrated in the ABAP dictionary. The CDS views are ...

- defined in a text-based editor (only in ADT)
- fully and solely managed by AS ABAP
- supported on any DB (native integration in HANA)
- consumed in Open SQL like SE11 views
- extensible on model level using modification-free view enhancements and on meta-model level using domain-specific annotations

Core Data Services in ABAP

Code-to-Data paradigm

- Supported through extended view functionality

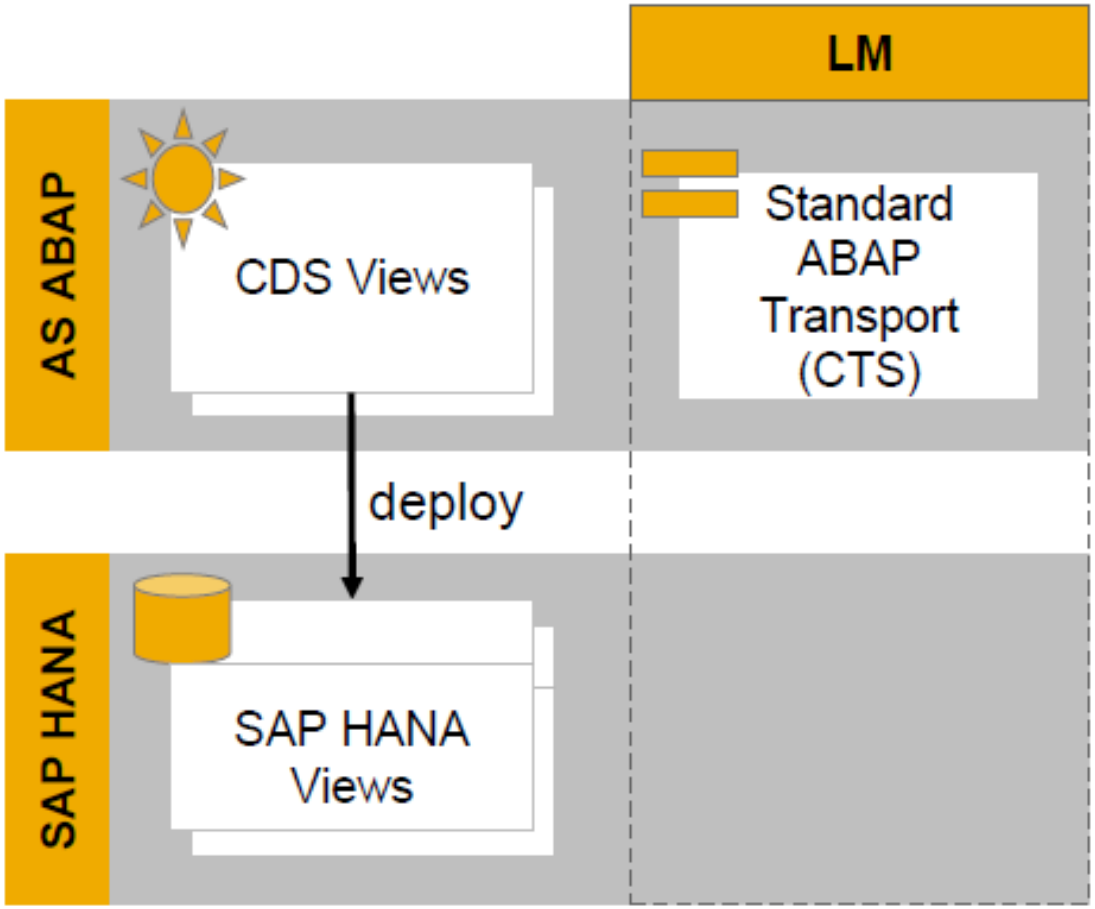
Definition of semantically rich data models in the ABAP Dictionary

- 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 Views vs. Open SQL

~~Competitors?~~

Not at all!

Use "plain" Open SQL if you

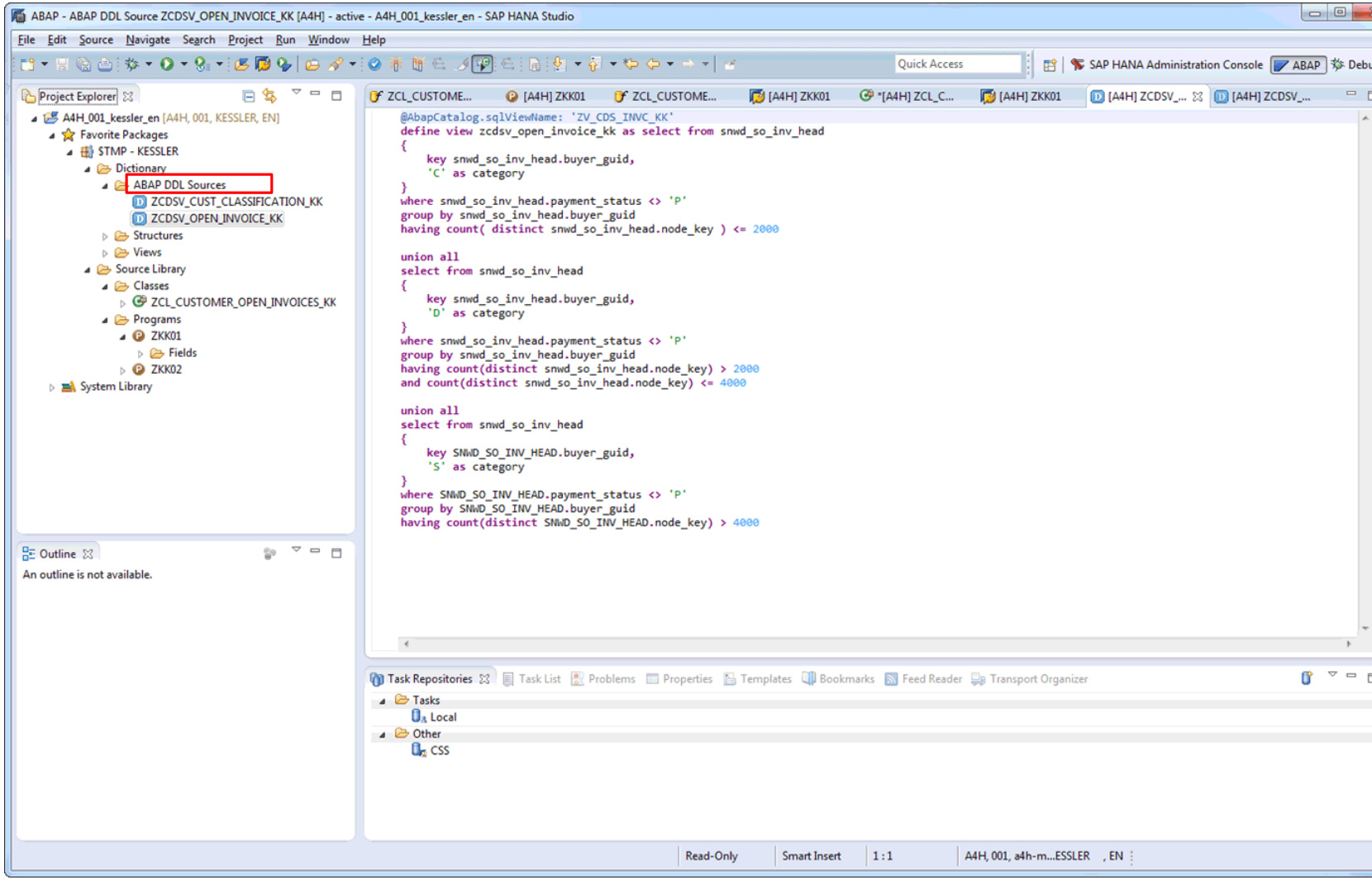
- … only need the query **"once"**, that is, in the coding (which might still be executed several times)
- … need features only available in Open SQL like `FOR ALL ENTRIES`

Use CDS views if you

- … have a real **re-use** case (same argument as for "old" dictionary views)
- … need features currently only available in DDL sources like `UNION`, `UNION ALL`, and so on
- … want to use features you will learn about in the upcoming units

Simple CDS View

- CDS views are defined using the CDS editor in ABAP in Eclipse
- Definition in an ABAP DDL Source (R3TR DDLS)
- Definition only possible with ABAP Development Tools in Eclipse (**not** via transaction SE11)



Core Data Services: View Definition Projection List

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

```
@AbapCatalog.sqlViewName: 'ZDDLS_CDS_10'
define view zcdsv_aggregations as select
 from snwd_so as so
 inner join snwd_bpa as bpa
         on so.buyer_guid = bpa.node_key
{
  key so_id as customer_id,
  bpa.company_name,
  'Literal' as string_literal,
  42        as integer_literal,
  so.currency_code,
  sum( so.gross_amount ) as
total_gross_amount
}
group by
    bpa.bp_id,
    bpa.company_name,
    so.currency_code
having sum( so.gross_amount ) > 1000
```

ABAP CDS View: Projection List

- Arithmetic expression:
  - Supported operators: +, - , * and unary –
  - Complex expressions and bracketing of sub-expressions possible
- Type casting:
  - Different operand types supported: Literal, column, path expression, build-in function, arithmetic expression
  - Various data types in ABAP namespace supported
  - Result length determined at activation time
  - No nesting of CAST expressions

Alias names required for resulting columns

```
@AbapCatalog.sqlViewName: 'ZDDLS_CDS_11'
define view zcdsv_arithmetics
as select from snwd_so as so
inner join snwd_bpa as bpa
  on so.buyer_guid = bpa.node_key
{
  key bpa.bp_id as customer_id,
  bpa.company_name,
  so.currency_code,
  ( so.gross_amount - so.net_amount )
    as tax_amount,
  0.85 * cast( so.gross_amount as
abap.fltp )
    as reduced_gross_amount
}
```

Core Data Services: View Definition Projection List

Conditional Expressions

- Available CASE constructs
  - Simple CASE
  - Searched CASE
- CASE constructs can be nested “CASE-in-CASE“
- Coalesce expression
  - Syntax short form for a CASE expression with two arguments
  - Returns the first argument if the value is not NULL, otherwise the second argument is returned

```
@AbapCatalog.sqlViewName: 'ZDDLS_CDS_12'
define view zcdsv_cond_exp
as select from snwd_so as so
left outer join snwd_so_inv_head as inv_head
  on so.node_key = inv_head.so_guid
{
  key so.so_id,
 so.currency_code,
 so.gross_amount,
 case delivery_status
       when ' ' then 'OPEN'
       when 'D' then 'DELIVERED'
       else delivery_status
 end as delivery_status_long,

 case
   when so.gross_amount > 1000
     then 'High Volume Sales Order'
   else ' '
 end as high_volumne_text,

 coalesce( inv_head.payment_status,
           'Not yet invoiced') as payment_status
}
```

View on View and View Extensions

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
}
```

Extend existing/delivered CDS view with:

- Table column
- Arithmetic & CASE expressions
- Literals

Extension "technique":

- Append to base view

Not allowed on views including:

- Grouping – for example, aggregation
- UNION (ALL) statements

```
@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 with Input Parameters

CDS Views 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
- Parameter can be used in
  - the projection list as element or in arithmetic expressions
  - expressions in WHERE or HAVING clauses
  - expression in ON conditions of JOIN statements
  - –…

- **Not supported on all databases**
  → **DBSYS-dependent feature**

```
@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 with Input Parameters: Consumption (1)

Consumption in a CDS view

- Provide (mandatory) input parameter(s)

```
@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 with Input Parameters: Consumption (2)

Consumption via Open SQL

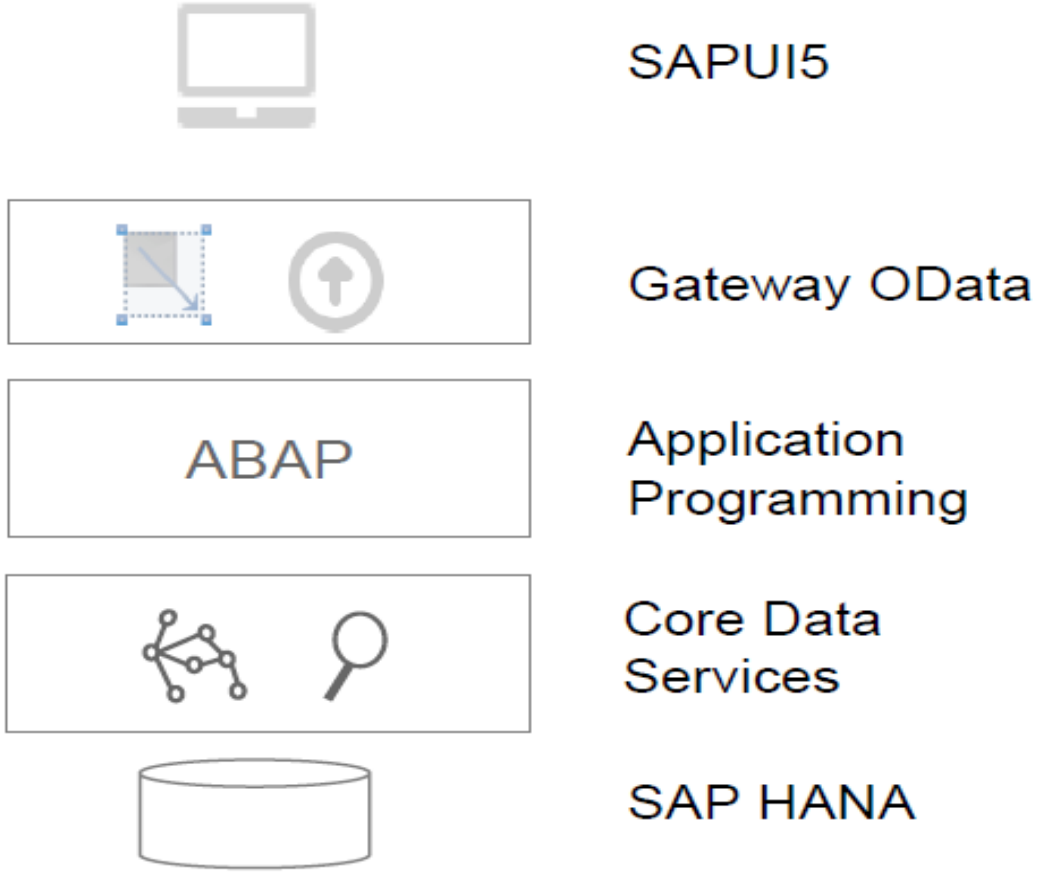
- Check if the feature is supported
- Provide (mandatory) input parameter(s)
- Suppress syntax warning using the pragma
- Provide a “fallback“ implementation / some error handling

```
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 ).
```

Consumption of CDS Views in SAP Gateway

CDS View Consumption

- Direct mapping of CDS views to an entity set in an OData service
- Completely modeled approach; no coding needed for read operations
- OData service can be easily bound to an SAPUI5 control



Additional Information

View Definition

- CDS View Definition Features
  - Projection List
  - Alias
- View-on-View Concept
- CDS View Extensions
- Annotations

CDS view with Associations

- CDS Views: UNION and JOIN Support
- CDS Associations:
  - Definition
  - Consumption
- Association Types
  - Ad-Hoc Association
  - Exposed Association
- Filtered Associations
- Advantages of Associations

ABAP Managed Database Procedures

Content

- **ABAP Managed Database Procedures**
  - Concept
  - Definition
  - Consumption
- **Exception Handling**
- **Additional Information**
  - Debugging AMDPs
  - AMDP Extension Concept

AMDP's – ABAP Managed Database Procedures

Advantages:

- Database procedures push down data-intensive logic to the database
- Reuse across multiple applications
- Provide the ability to perform integrity checks and grant privileges

ABAP Managed Database Procedures

Code-to-Data Paradigm

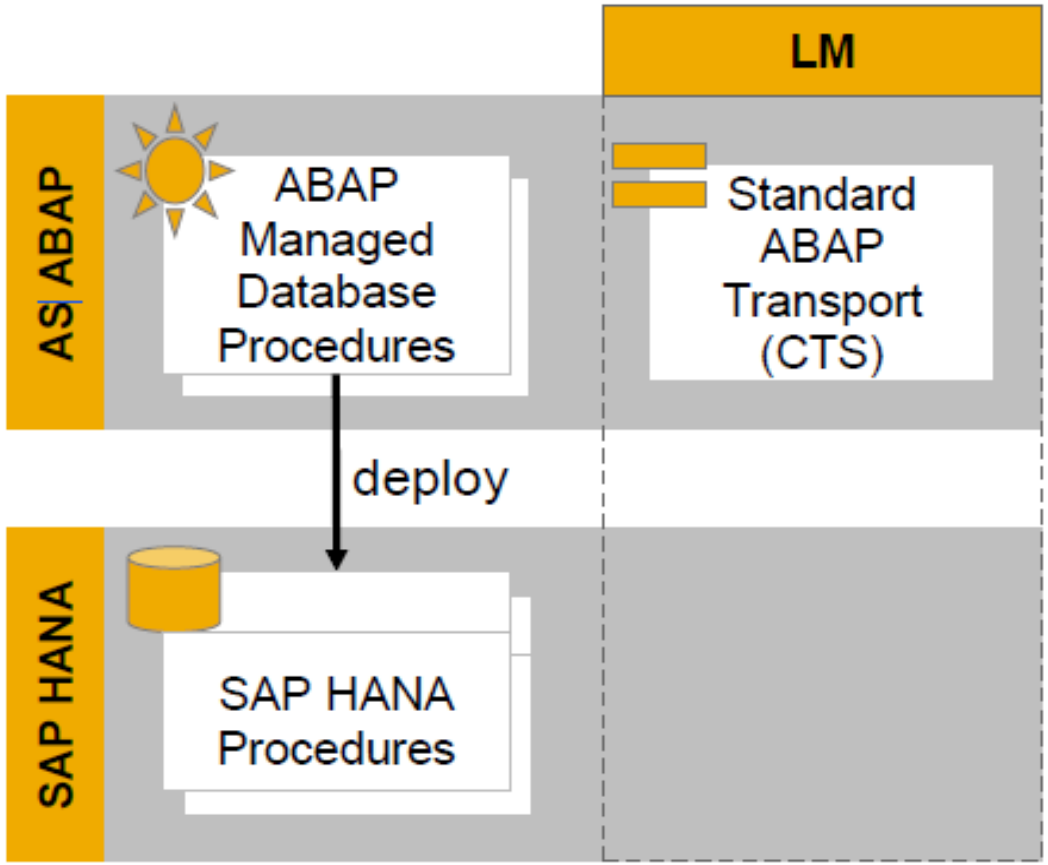
- Supported through embedding native database procedure coding

Definition & Consumption of AMDPs

- Definition / maintenance via ABAP Development Tools in Eclipse
- Standard ABAP class method as containers for database procedures coding Corresponding SAP HANA artifacts created automatically
- Consumption like any other ABAP class method

Fully integrated into the ABAP infrastructure

- Consistent lifecycle management with all other ABAP artifacts (only transport of ABAP objects required)
- Syntax check provided for SQLScript code
- Detailed analysis of ABAP runtime errors



Simplified control using AMDP

- Defining an AMDP
- Implementation
- Consumption
- Debugging an AMDP
- Enhancements for AMDPs

Code-To-Data paradigm for AMDP's

- AMDPs allow developers to create and manage stored procedures directly in ABAP while they are executed on the database level
- AMDPs embedded in the ABAP execution means that they are fully managed by the SAP system's CTS, and behave like any other ABAP artifact from a lifecycle management perspective.
- Though AMDPs are defined on the ABAP level, they are by nature dependent on the underlying database platform, so that they can fully optimize the database in use
- For this reason, the implementation language will vary depending on the database in use. For SAP HANA, the implementation language is SQLScript, so creating an AMDP for SAP HANA is as simple as implementing SQLScript in your ABAP program, both in new and existing programs.

Class Definition

Prerequisites in class definition

- Classes with AMDPs must use interface IF_AMDP_MARKER_HDB
- All AMDP method parameters must be passed by value (like RFC)
- All AMDP parameters must be tables with elementary components or scalar types
- AMDPs support (secondary) database connections to the primary database via input parameter **CONNECTION** (type DBCON_NAME)

```
CLASS zcl_amdp_simple_00 DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.
    INTERFACES: if_amdp_marker_hdb.


    METHODS get_customer_infos
      IMPORTING
        VALUE(<input>) TYPE <input_type>
      EXPORTING
        VALUE(<output>) TYPE <output_type>
      RAISING <amdp_exception> .
ENDCLASS.
```

Class Implementation

Extended method implementation syntax: BY DATABASE PROCEDURE ...

- Indicates method body contains database-specific code not executable on the ABAP server
- Database platform (currently only SAP HANA supported)
- Database procedure language (for example SQLScript)
- Used ABAP Dictionary tables, Dictionary views, and other AMDP methods
- Native SAP HANA SQLScript source code

```
METHOD get_customer_infos
  BY DATABASE PROCEDURE
  FOR HDB
  LANGUAGE SQLSCRIPT
  OPTIONS READ-ONLY
  USING <dictionary_artifacts>.

  --meaningful SQLScript coding

  et_customer_info =
   SELECT ... FROM ...;

ENDMETHOD.
```

Consumption & Artifact Creation

AMDP consumption like any other ABAP method call

AMDP Runtime:

- At first call of an AMDP, several SAP HANA artifacts are created in the SAP<SID> schema, such as the SAP HANA database procedure
- Artifact creation can alternatively been triggered via ABAP report **RSDBGEN_AMDP**
- When an AMDP is processed, the ABAP stack calls the corresponding database procedure in SAP HANA

```
REPORT zr_amdp_01_simple_call.

DATA(lo_amdp) = NEW zcl_amdp_simple_00( ).

lo_amdp->get_customer_infos(
      IMPORTING
        et_customer_info = DATA(lt_result)
      ).
```

Exception Handling

Catchable Exceptions

- Several AMDP runtime errors have a corresponding (catchable) exception
- Naming convention:

<ERROR_NAME> □ CX_<ERROR_NAME>

- To-Dos for AMDP

  Developers/Consumers:

  — Add RAISING clause to the AMDP method definition

  — Enclose the AMDP call in a TRY… CATCH block

```
"definition
METHODS <method_name>
        <method_interface>
      RAISING cx_amdp_error.
...

"consumption
TRY.
    <method_call>

  CATCH cx_amdp_execution_failed INTO DATA(lx).
    "do some meaningful error handling
ENDTRY.
```

Additional Information

- Extension Concept
  - Extensibility of AMDP's supported via AMDP BADI's
- Debugging of AMDP's
  - Fully integrated support in ABAP development tools in Eclipse is not yet available
  - Workaround : External session debugging

Extensibility: AMDP Business Add-Ins (BAdIs)

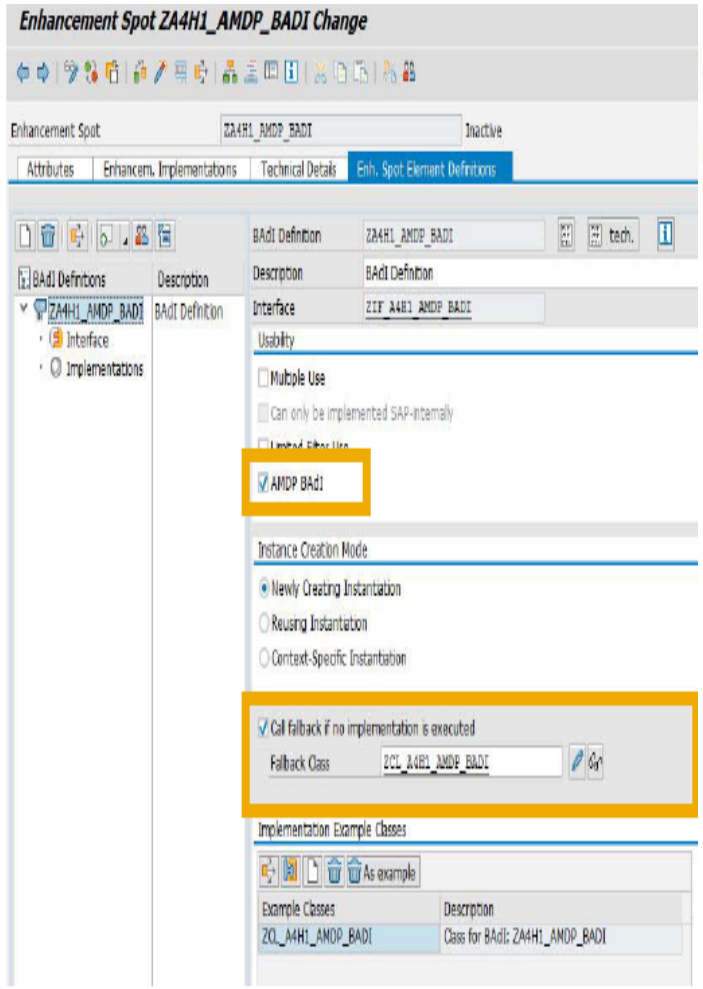
Modification-free enhancement of AMDP methods using BAdIs and calling points

Special BAdIs provided (AMDP BAdIs)

- AMDP BAdI interface has to include the tag interfaces **IF_AMDP_MARKER_HDB**
- Only AMDP methods allowed
- Fallback method is mandatory

Integrated in the Switch & Enhancement Framework

Filters are currently not supported

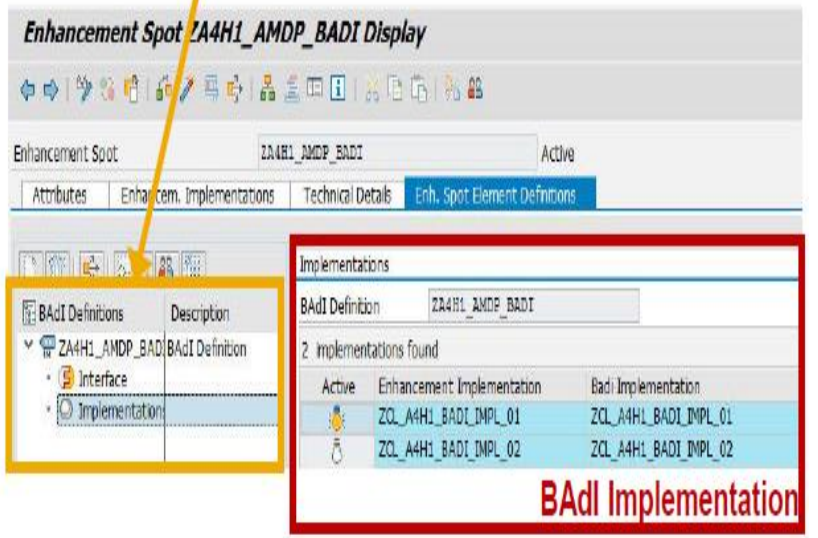


Consumption

- Call of AMDP BAdI like any other AMDP call
  - Inclusion in USING list
  - Consumption via CALL <AMDP_NAME>

BAdI Caller

```
CLASS zcl_a4h1_call_amdp_badi IMPLEMENTATION.
  METHOD call_badi
    by DATABASE PROCEDURE
    FOR HDB
    LANGUAGE SQLSCRIPT
    options read-only
    using ZA4H1_AMDP_BADI=>get_customer_infos.

   call "ZA4H1_AMDP_BADI=>GET_CUSTOMER_INFOS"( et_customer_info => et_customer_info );
  ENDMETHOD.
```



Reference Links

- **More information**
- **On AMDPs:** http://scn.sap.com/docs/DOC-51146
- **On SQLScript:**http://scn.sap.com/community/abap/hana/blog/2014/09/26/code-push-down-for-hana-from-abap-starts-with-open-sql
- SCN blog: http://scn.sap.com/community/abap/blog/2014/02/06/abap-news-for-release-740-sp05
- http://scn.sap.com/community/abap/hana/blog/2014/02/03/abap-for-hana-code-push-down
- **For CDS Views:**
- http://scn.sap.com/community/abap/eclipse/blog/2014/02/04/new-data-modeling-features-in-abap-for-hana
- **For ABAP on HANA**
- **http://scn.sap.com/community/abap/hana**
- **http://scn.sap.com/community/abap/eclipse**.

TATA CONSULTANCY SERVICES
Experience certainty.

Thank You!!!