Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

SAP ABAP ON HANA

Overview

AB1011 - ABAP ON HANA

HANA – **H**igh-performance **AN**alytic **A**ppliance

Lesson Objectives

- **On Completing this course, participants will be able to**

  - Understand SAP HANA Project Creation
  - Understand ABAP on HANA Native and Open SQL
  - Understand Optimizing ABAP for SAP HANA
  - To know Code Pushdown with Open SQL
  - Understand Code Comparison
  - Understand SQL Clause New Features

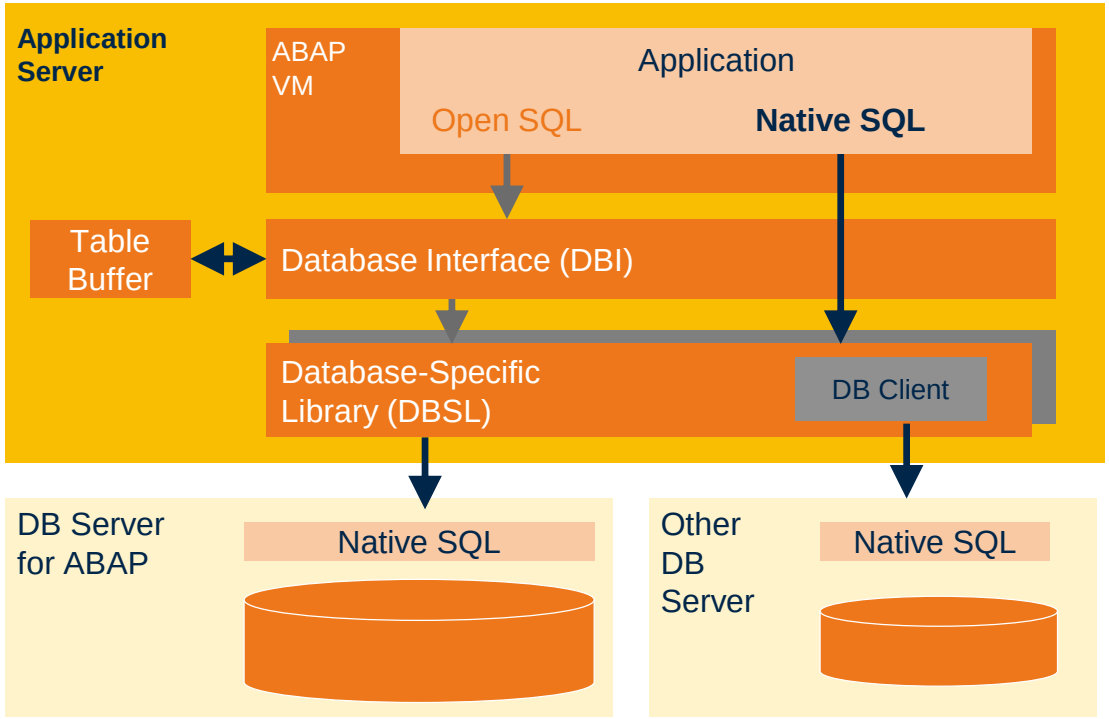
Contents

Contents

ABAP on HANA : Native and Open SQL

Native SQL in ABAP

- Native SQL in a nutshell
- Native SQL are the queries that are specific to a particular dbase used.
- Native SQL is used in ADBC.

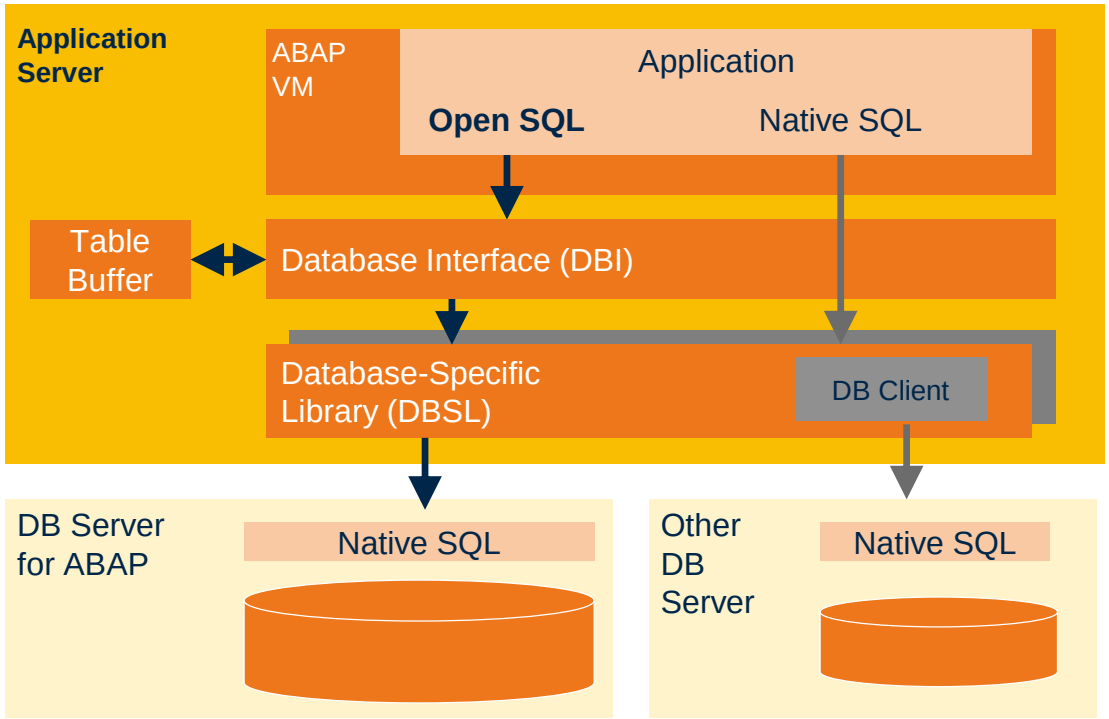


Native SQL Pitfalls

- Loosely integrated into ABAP
  - No syntax check at compile time. Statements are directly sent to the database system. Handle exception CX_SQL_EXCEPTION
  - No automatic client handling, no table buffering.
  - All tables, in all schemas can be accessed
- Developer responsible for
  - Client handling, accessing correct schema
  - Releasing DB resources
  - Proper locking and commit handling

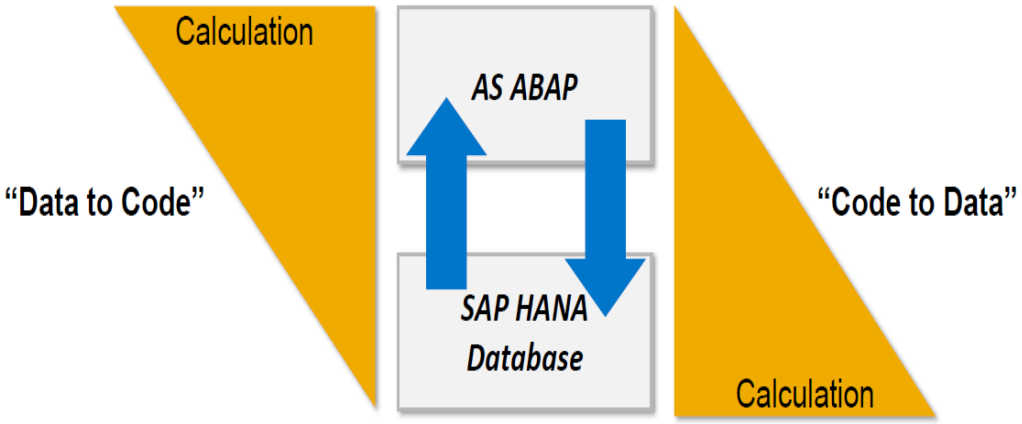
What Is Open SQL?

- Open SQL consists of a set of ABAP statements that perform operations on the central database in the SAP Web AS ABAP.
- Provides a uniform syntax and semantics for all of the database systems supported by SAP.
- Can only work with database tables that have been created in the ABAP Dictionary.



Open SQL Supports Code Push down

- ✓ Push down data intense computations and calculations to the HANA DB layer
- ✓ Avoid bringing all the data to the ABAP layer.



Advanced Open SQL Less restrictions and more freedom

- Support more standard SQL features (SQL92) in Open SQL
- Limitations removed starting with ABAP 7.4 SP05
- For SAP HANA and other database platforms

Restrictions in ABAP < 7.4 SP05

Advanced Open SQL Features overview - SP05

- New syntax defined for new features
  - Comma separated select list
  - Escaping of host variables with “@”
- SQL Expressions as column information after SELECT
  - Arithmetical Expressions
  - String Expressions
  - Conditional Expressions (CASE)
- Switching off the automatic client handling with the USING CLIENT clause
- Maximal number of sub-queries increased from 9 to 50
- RIGHT OUTER JOIN now supported
- Enhanced bracketing in JOIN expressions
- New functionality in ON conditions of JOIN expressions, e.g.
  - Use of BETWEEN or “>” for comparisons
  - Possibility to use fields of the right table in the WHERE clause of LEFT OUTER JOINs
- Maximal number of tables in JOINs increased to 50!
- Access to ABAP Core Data Services views

Advanced Open SQL Features overview - SP08

- New column specification data_source~* after SELECT statement
- Inline declarations for target range of SELECT statement
- New SQL expressions
- Consumption of parameterized CDS views
- Arrangement of the INTO clause
- Stricter checks for syntax rules
- Strict mode in the syntax check

- More Details: http://help.sap.com/abapdocu_740/en/index.htm?file=ABENNEWS-740_SP05-OPEN_SQL.htm

Advanced Open SQL Feature details: Strict mode in the syntax checks

@ - Escaping

Implies

Comma separation

Implies

Strict mode
(Nearly) all warnings become syntax errors

Note: Old syntax still supported; New syntax only obligatory when using new features

Don’t Code Like This!

```
SELECT * FROM snwd_bpa INTO ls_bpa.
 SELECT * FROM snwd_so INTO ls_soi
        WHERE buyer_guid = ls_bpa-node_key.
**  Do something ...
   SELECT * FROM snwd_so_i INTO TABLE lt_soi
         WHERE parent_key = ls_soi-node_key.
**  Do something ...
   LOOP AT lt_soi INTO ls_soi.
     SELECT SINGLE FROM snwd_pd into ls_pd
          WHERE node_key = ls_soi-product_guid.
**    Do something ...
   ENDLOOP.
 ENDSELECT.
ENDSELECT.
```

X

- Nested selects & loops result in lots of data packages and identical DB accesses
- SELECT * often reads far more than needed → large data transfer effort on DB and Network

Open SQL Supports Aggregation, Joins And Sub-Queries

```
SELECT COUNT(*) SUM( gross_amount )
    MAX( gross_amount ) MIN( gross_amount) …
 FROM snwd_so INTO … GROUP BY …
```

```
SELECT bp~bp_id AS bp_id
    so~created_at AS created_at
 FROM snwd_so AS so INNER JOIN snwd_bpa AS bp
    ON so~buyer_guid = bp~node_key …
```

```
SELECT bp_id FROM snwd_bpa INTO …
 WHERE node_key IN
          ( SELECT buyer_guid FROM snwd_so
            WHERE gross_amount =

( SELECT MAX( gross_amount ) FROM snwd_so ) )
.
```

- Open SQL allows calculating aggregate values – including grouping by non-aggregated fields
  .
- Open SQL supports INNER and OUTER joins,
  e.g. read customer & sales order data in one go.
- Open SQL supports sub-queries,
  e.g. determine customers with the largest sales orders

Optimizing ABAP for SAP HANA

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

In addition to above mentioned 5 golden rules, Implement Code Pushdown approach for data-intensive calculations to benefit from SAP HANA.

Conclusion II: Some Guidelines Become More Important

These guidelines are even more important

Keep result sets small

Avoid unpacking columns unnecessarily

- Select as few fields as possible, especially from tables in the column store

Minimize number of database accesses

Avoid unpacking same columns/tables unnecessarily

- Use mass processing wherever possible
- Select all rows and columns in one SQL statement

Conclusion III: Some Guidelines Changed

These guidelines changed

Minimize search overhead

WHERE clauses using non-indexed fields are not so bad anymore

- with in-memory technology, full table scans are fast
- only a few indexes exist on SAP HANA

Keep unnecessary load away from DB

Push data-intensive calculations to SAP HANA

- Calculation and aggregation are very efficient on SAP HANA
- Significant reduction of data transfer adds to the performance gain

“Code pushdown” Begins with Open SQL

- ✓ Use aggregate functions where relevant instead of doing the aggregations in the ABAP layer.
- ✓ Use arithmetic and string expressions within Open SQL statements.
- ✓ 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.

Code Comparison

Old Code

```
SELECT so_id
       currency_code
       gross_amount
  FROM snwd_so
  INTO TABLE lt_so_amount.
```

New Code

```
SELECT so_id,
       currency_code,
       gross_amount
  FROM snwd_so DATA
  INTO TABLE @lt_so_amount.
```

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).
```

SQL Clause New Features

New Open SQL Syntax

- Escaping of host variables
- Comma separated element list

Target Type Inference

New SELECT List Features

- Aggregation functions
- Literal values (next slide)
- Arithmetic expressions (next slide)
- String expression (next slide)

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

```
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.
```

Literal Values

Literal Values

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

Literal Values

```
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.
```

Arithmetic 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).
```

Open SQL Aggregation

```
REPORT zr_opensql_01_aggregation.

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.

cl_demo_output=>display_data( value = lt_result )
```

Open SQL CASE Statement

```
"searched case - more advanced
DATA lv_discount TYPE p LENGTH 1 DECIMALS 1 VALUE '0.8'.
SELECT so_id,
       company_name,
       gross_amount AS orig_amount,
      CASE WHEN company_name = 'SAP'
        THEN ( gross_amount * @lv_discount )
        ELSE gross_amount
      END AS discount_amount
 FROM snwd_so AS so
 INNER JOIN snwd_bpa AS bpa
         ON so~buyer_guid = bpa~node_key
 INTO TABLE @DATA(lt_adv_example_searched_case).
```

Conditional Expressions

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).
```

GROUP BY & HAVING Clauses

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.
```

JOIN Statements

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).
```

Summary

- By end of this course, participants know
  - Understand ABAP on HANA Native and Open SQL
  - To work with Code Pushdown with Open SQL
  - Understand Code Comparison
  - To know Automatic Client Handling
  - How to write SQL Clause New Features
  - How to write Literal Values and Arithmetic Expressions
  - How to write Open SQL Aggregation and CASE Statement
  - How to write Conditional Expressions and GROUP BY & HAVING Clauses
  - How to write JOIN Statements

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Thank you