ASUG ANNUAL CONFERENCE

SAP HANA SQLScript Basics, Debugging, and ABAP Connectivity

Rich Heilman, SAP HANA Product Management, SAP Labs, LLC.

Disclaimer

This presentation outlines our general product direction and should not be relied on in making a purchase decision. This presentation is not subject to your license agreement or any other agreement with SAP. SAP has no obligation to pursue any course of business outlined in this presentation or to develop or release any functionality mentioned in this presentation. This presentation and SAP's strategy and possible future developments are subject to change and may be changed by SAP at any time for any reason without notice. This document is provided without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. SAP assumes no responsibility for errors or omissions in this document, except if such damages were caused by SAP intentionally or grossly negligent.

Agenda

Introduction to SQLScript Basics & Debugging

Demo

ABAP for HANA Connectivity

Demo

SQLScript Basics & Debugging

SQLScript
What?

SQL Script is an interface for applications to access SAP HANA

Extension of ANSI Standard SQL

Language for creating stored procedures in HANA

- Declarative Logic including SELECT queries, Built-In Calculation Engine functions
- Orchestration Logic including Data Definition Language(DDL), Data Manipulation Language(DML), assignment, imperative logic

SQLScript

Why?

The main goal of SQLScript is to allow the execution of data intensive calculations inside SAP HANA

There are two reasons why this is required to achieve the best performance:

- Eliminates the need to transfer large amounts of data from the database to the application
- Calculations need to be executed in the database layer to get the maximum benefit from SAP HANA features such as fast column operations, query optimization and parallel execution. If applications fetch data as sets of rows for processing on application level they will not benefit from these features

SQLScript

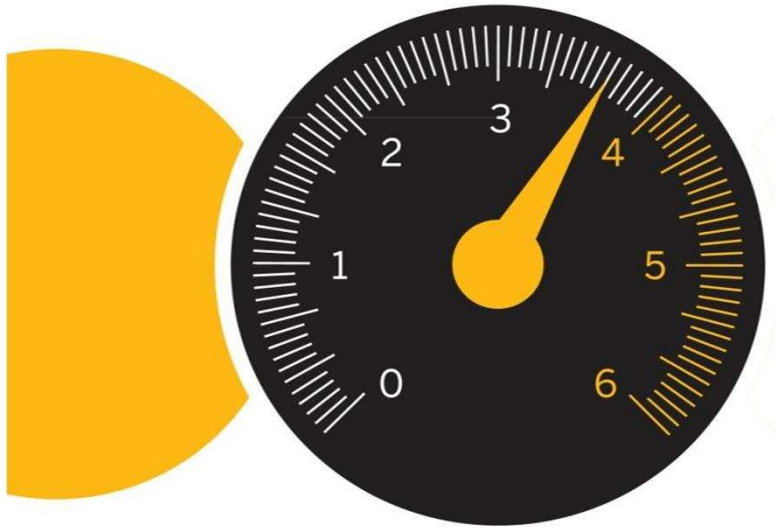
Advantages

Compared to plain SQL queries, SQL Script has the following advantages:

- Functions can return multiple results, while a SQL query returns only one result set
- Complex functions can be broken down into smaller functions. Enables modular programming, reuse and a better understandability by functional abstraction. For structuring complex queries, standard SQL only allows the definition of SQL views. However, SQL views have no parameters
- SQLScript supports local variables for intermediate results with implicitly defined types. With standard SQL, it would be required to define globally visible views even for intermediate steps
- SQL Script has control logic such as if/else that is not available in SQL

SQLScript

Performance gains



Presentation logic

Control flow logic

HANA

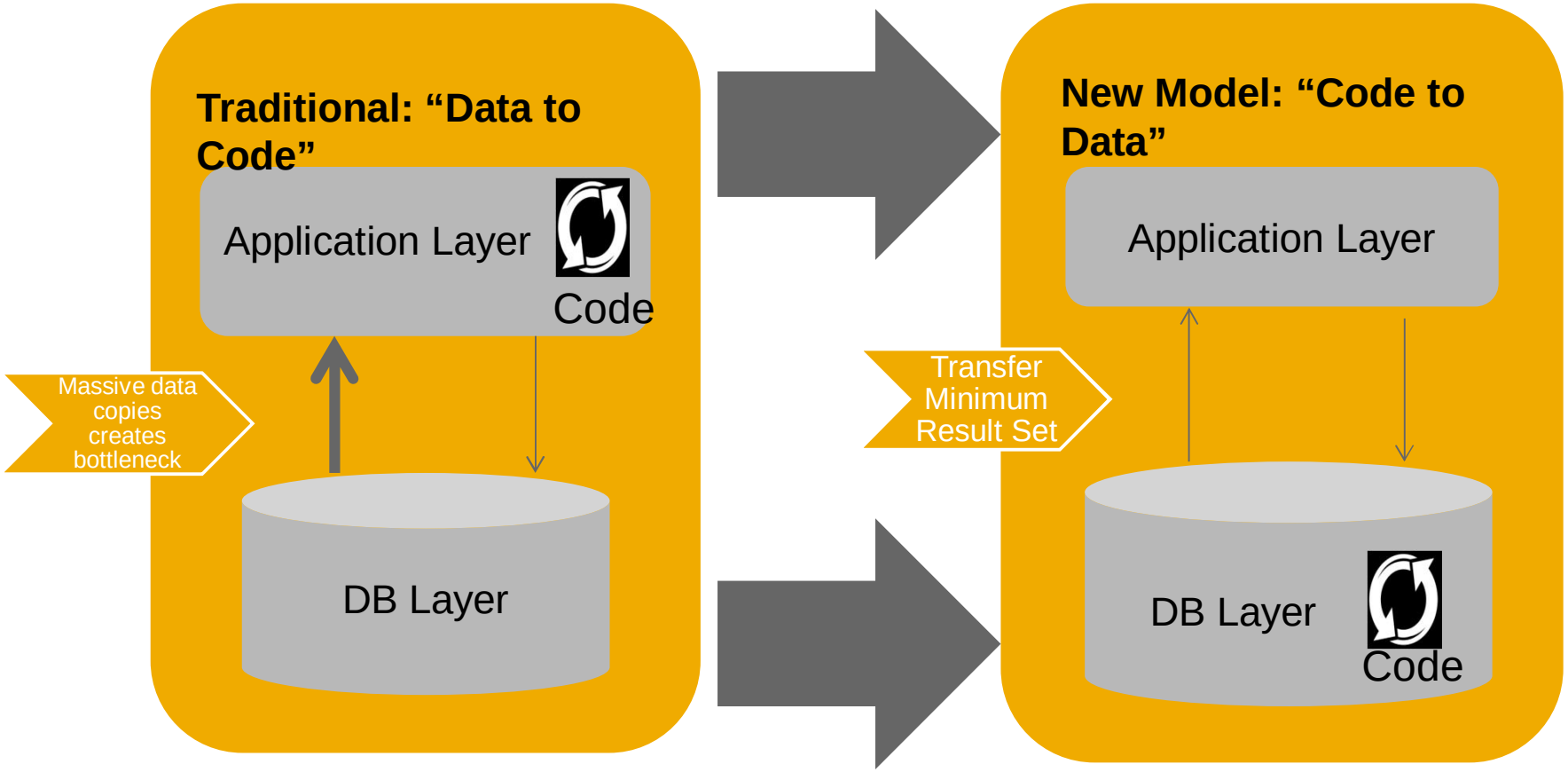
Data

Calculation logic

SQLScript

SQLScript

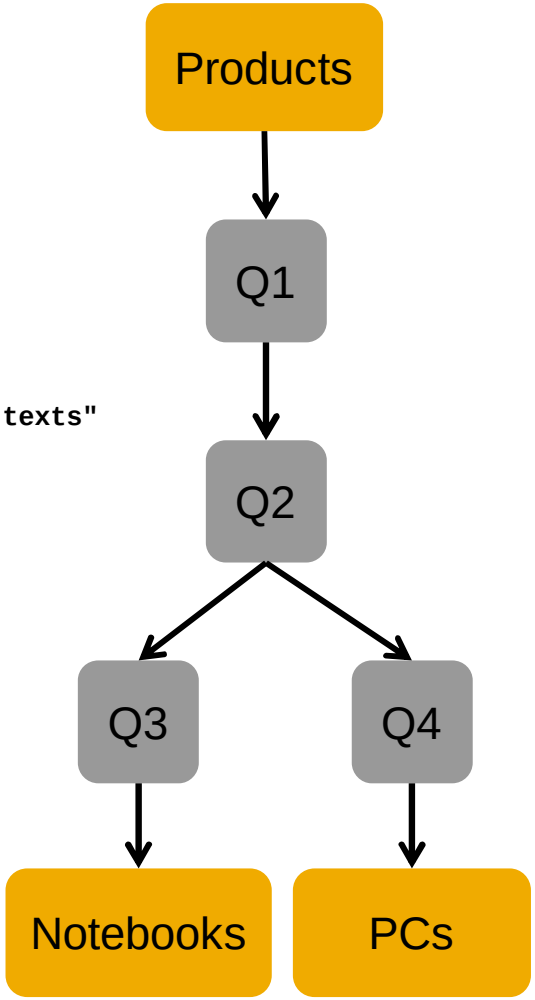
Traditional model vs. new model



SQLScript

Code example

```
BEGIN
...

-- Query 1
product_ids = select "ProductId", "Category", "DescId"
        from "SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm.data::products"
                    where "Category" = 'Notebooks'
                        or "Category" = 'PC';

-- Query 2
product_texts = select "ProductId", "Category", "DescId", "Text"
         from :product_ids as prod_ids
            inner join "SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm.data::texts"
                 as texts on prod_ids."DescId" = texts."TextId";

-- Query 3
out_notebook_count = select count(*) as cnt from
                      :product_texts where "Category" = 'Notebooks';

-- Query 4
out_pc_count = select count(*) as cnt from
                      :product_texts where "Category" = 'PC';

...

END;
```



SQLScript

Parallel processing

SELECT statements are executed in parallel unless:

- Any local scalar parameters and variables are used in the procedure
- Read/Write procedures or DML/DDL operations are executed
- Imperative logic is used within the procedure
- Any SQL statements are used that are not assigned to a variable

SQLScript

CE(Calculation Engine) Built in functions

	SQL	CE-Built In Function
SELECT on column table	out = SELECT A, B, C from "COLUMN_TABLE"	out = CE_COLUMN_TABLE("COLUMN_TABLE", [A, B, C])
SELECT on attribute view	out = SELECT A, B, C from "ATTRIBUTE_VIEW"	out = CE_JOIN_VIEW("ATTRIBUTE_VIEW", [A, B, C])
SELECT on olap view	out = SELECT A, B, C, SUM(D) from "ANALYTIC_VIEW" GROUP BY A, B, C	out = CE_OLAP_VIEW("ANALYTIC_VIEW", [A, B, C]);
WHERE HAVING	out = SELECT A, B, C, SUM(D) from "ANALYTIC_VIEW" WHERE B = 'value' AND C = 'value'	col_tab= CE_COLUMN_TABLE("COLUMN_TABLE"); out = CE_PROJECTION(col_tab, [A, B, C], ' "B" = ''value'' AND "C" = ''value'' ');
GROUP BY	out = SELECT A, B, C, SUM(D) FROM"COLUMN_TABLE" GROUP BY A, B, C	col_tab= CE_COLUMN_TABLE("COLUMN_TABLE"); out = CE_AGGREGATION((col_tab, SUM(D), [A, B, C]);
INNER JOIN	out = SELECT A, B, Y, SUM(D) from "COLTAB1" INNER JOIN "COLTAB2" WHERE "COLTAB1"."KEY1" = "COLTAB2"."KEY1" AND "COLTAB1"."KEY2" = "COLTAB2"."KEY2"	out = CE_JOIN("COLTAB1","COLTAB2", [KEY1, KEY2], [A, B, Y, D])
LEFT OUTER JOIN	out = SELECT A, B, Y, SUM(D) from "COLTAB1" LEFT OUTER JOIN "COLTAB2" WHERE "COLTAB1"."KEY1" = "COLTAB2"."KEY1" AND "COLTAB1"."KEY2" = "COLTAB2"."KEY2"	out = CE_LEFT_OUTER_JOIN("COLTAB1","COLTAB2", [KEY1, KEY2], [A, B, Y, D])

SQLScript

Built in function code example

Built in functions should be used exclusively where possible

Calculation Engine functions should not be mixed with standard SQL statements

Queries can be well optimized and parallelized by the engine

```
bp_addresses =
    select a."PartnerId", a."PartnerRole", a."EmailAddress", a."CompanyName",
        a."AddressId", b."City", b."PostalCode", b."Street"
         from "SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm.data::businessPartner" as a
          inner join "SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm.data::addresses" as b
                  on a."AddressId" = b."AddressId" where a."PartnerRole" = :partnerrole;
```

SQL

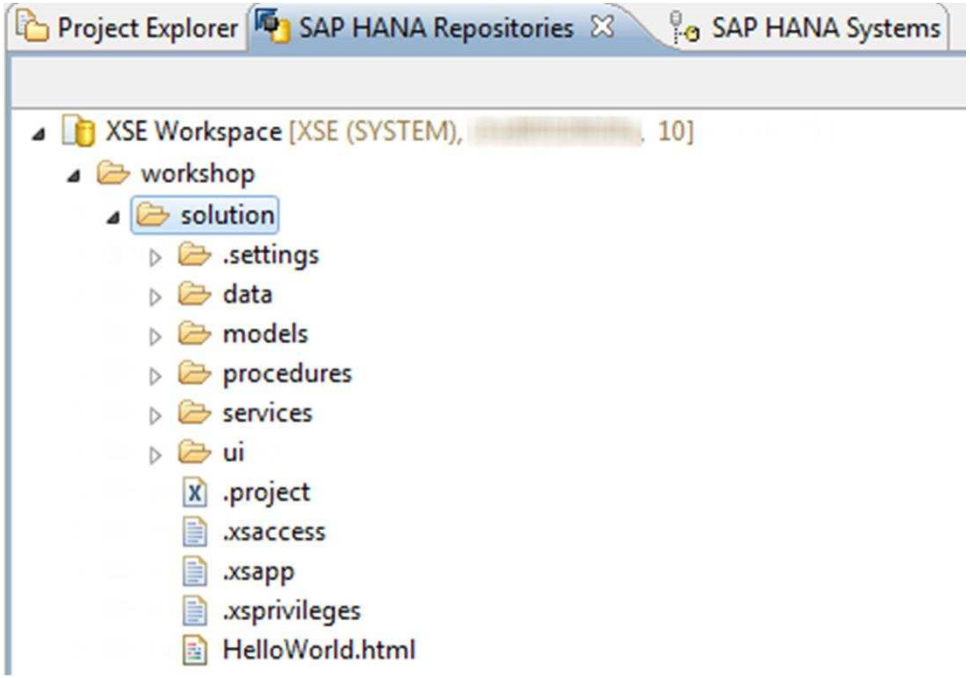
```
lt_bp = CE_COLUMN_TABLE("SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm.data::businessPartner",
        ["PartnerId", "PartnerRole", "EmailAddress", "CompanyName", "AddressId" ]);

lt_bp_proj = CE_PROJECTION(:lt_bp, ["PartnerId", "PartnerRole", "EmailAddress" ,
               "CompanyName", "AddressId" ], '"PartnerRole" = :partnerrole' );

lt_address = CE_COLUMN_TABLE("SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm.data::addresses",
                 ["AddressId", "City", "PostalCode", "Street"]);

bp_addresses = CE_JOIN(:lt_bp_proj, :lt_address, ["AddressId"],
                 ["PartnerId", "PartnerRole", "EmailAddress", "CompanyName",
                  "AddressId", "City", "PostalCode", "Street"]);
```

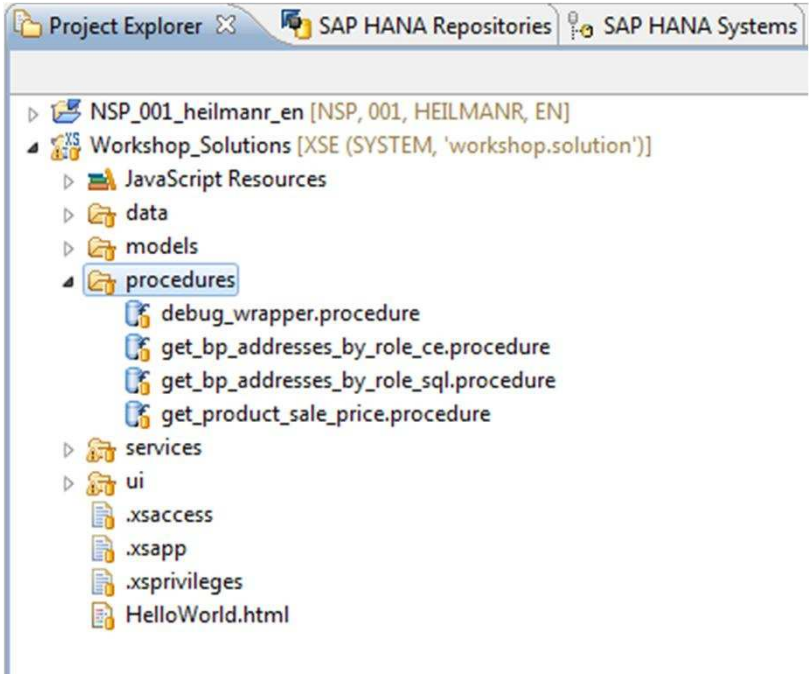
CE Function

SQLScript

New SQLScript Editor

Available since SP05
Project based approach
Client side syntax checking
Code hints
Syntax highlighting
Local table types





SQLScript

New SQLScript Editor

get_bp_addresses_by_role_sql.procedure

XSE (SYSTEM) chid00520626a 10

SQLScript | Local Table Types

```
/*
You can create new table types and use them as in
The table types must be defined using SQL syntax,
CREATE TYPE <Type Name> AS TABLE (<filed Name1> <
*/
create type tt_bp_addresses as table (
  PartnerId nvarchar(10),
  PartnerRole nvarchar(3),
  EmailAddress nvarchar(255),
  CompanyName nvarchar(80),
  AddressId nvarchar(10),
  City nvarchar(40),
  PostalCode nvarchar(10),
  Street nvarchar(60)
)
```

get_bp_addresses_by_role_sql.procedure

XSE (SYSTEM) chid00520626a 10

SQLScript | Local Table Types

```
CREATE PROCEDURE get_bp_addresses_by_role_sql ( in partnerrole nvarchar(3),
                                                out bp_addresses tt_bp_addresses)
    LANGUAGE SQLSCRIPT
    SQL SECURITY INVOKER
    READS SQL DATA AS
BEGIN
/****************************
    Write your procedure logic
****************************/
bp_addresses =
     select a."PartnerId", a."PartnerRole", a."EmailAddress", a."CompanyName",
         a."AddressId", b."City", b."PostalCode", b."Street"
          from "SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm.data::businessPartner" as a
           inner join "SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm.data::addresses" as b
                   on a."AddressId" = b."AddressId"
                         where a."PartnerRole" = :partnerrole;
END;
```

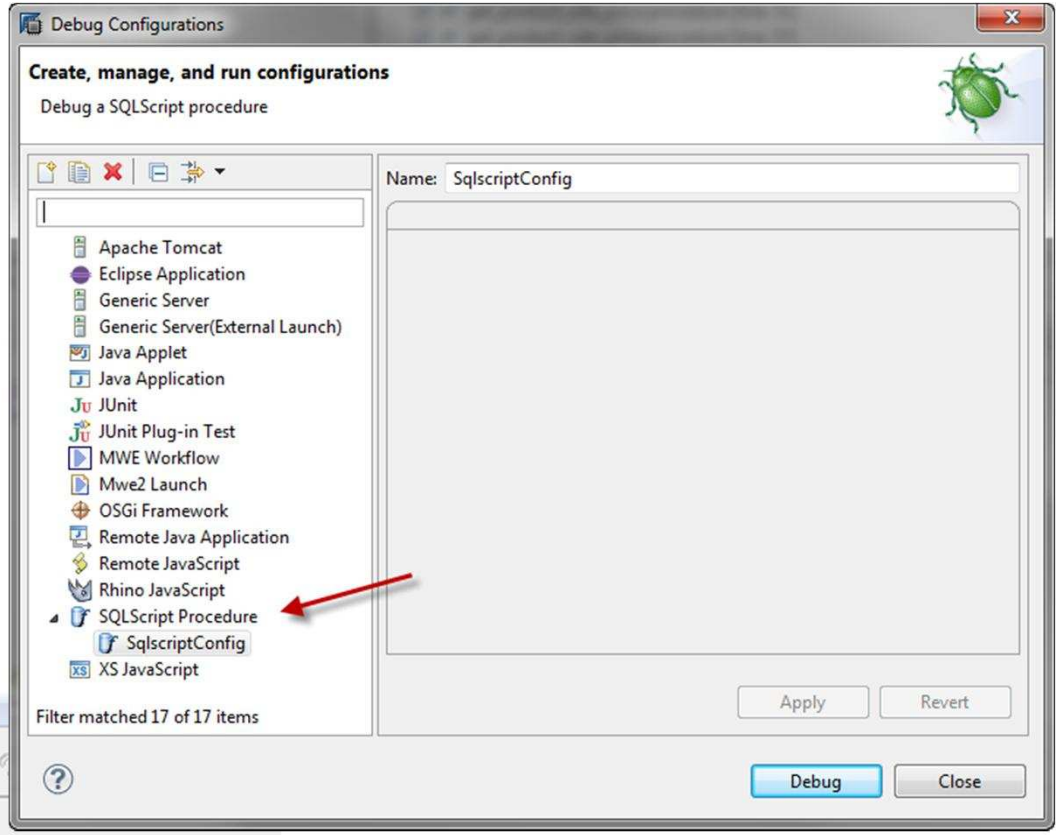
SQLScript
Debugging

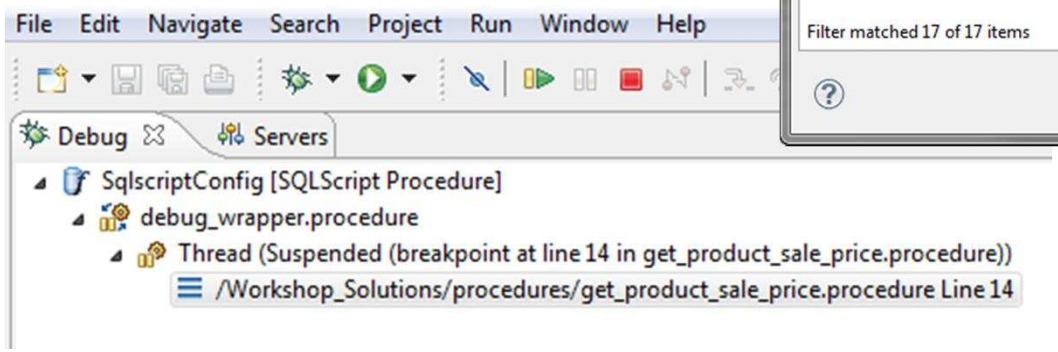
Available since SP05

Debug session management

Breakpoint management

Variable and parameter evaluation





SQLScript Debugger

Debug Perspective

File Edit Navigate Search Project Run Window Help

Debug | Servers

- SqlscriptConfig [SQLScript Procedure]
  - debug_wrapper.procedure
    - Thread (Suspended (breakpoint at line 32 in get_product_sale_price.procedure))
      - /Workshop_Solutions/procedures/get_product_sale_price.procedure Line 32

Variables | Breakpoints

Name	Value
LV_CATEGORY	Notebook
LV_DISCOUNT	0.20
PRODUCTID	HT-10
PRODUCT_SALE_PRICE	0 Rows
LT_PRODUCT	1 Rows

Variable Evaluation

debug_wrapper.procedure | get_product_sale_price.procedure

XSE (SYSTEM) chid00520626a 10

SQLScript | Local Table Types

```
if :lv_category = 'Notebooks' then
     lv_discount := .20;
elseif :lv_category = 'Handhelds' then
     lv_discount := .25;
elseif :lv_category = 'Flat screens' then
     lv_discount := .30;
elseif :lv_category like '%printers%' then
     lv_discount := .30;
else
     lv_discount := 1.00;  -- No discount
end if;

product_sale_price =
        select "ProductId", "Category", "Price",
             "Price" - cast(("Price" * :lv_discount) as decimal(15,2)) as "SalePrice"
                        from :lt_product;

END;
```

Set Breakpoints

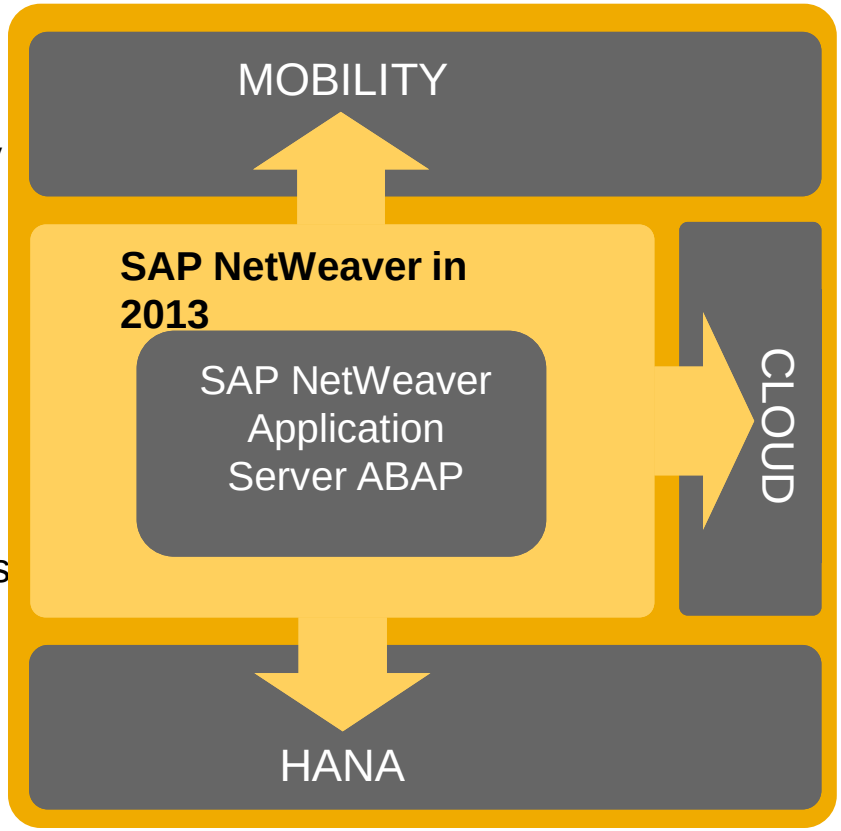
Demo

ABAP for SAP HANA

SAP NetWeaver Application Server ABAP

Empowering classic & new SAP products

- 50.000+ customers of ABAP-based products
- Millions of ABAP developers, SCN as community
- A thriving partner ecosystem
- Proven, robust and scalable
- Extends into HANA, Mobility and Cloud
- Evolves continuously w/o disruption
- Enables hybrid on-premise/on-demand scenarios



SAP NetWeaver Application Server ABAP, a strong pillar in SAP's product strategy

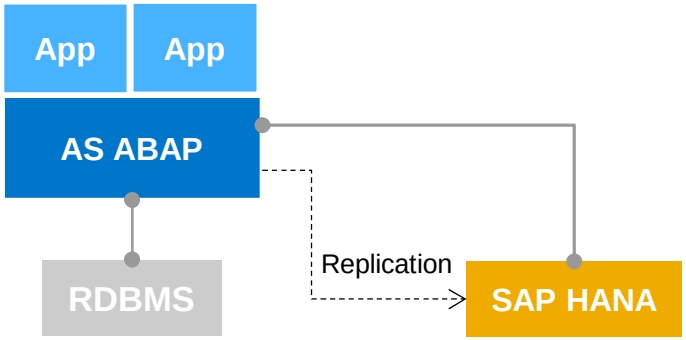
Stepwise Adoption of SAP HANA by the Application Server ABAP

Step 1	Step 2	Step 3
ABAP can access SAP HANA	**ABAP runs on SAP HANA**	**ABAP optimized for SAP HANA**
Side-by-side scenarios on **AS ABAP 7.X**	SAP NetWeaver BW on **AS ABAP 7.3**	**AS ABAP 7.4** (Future/Outlook)

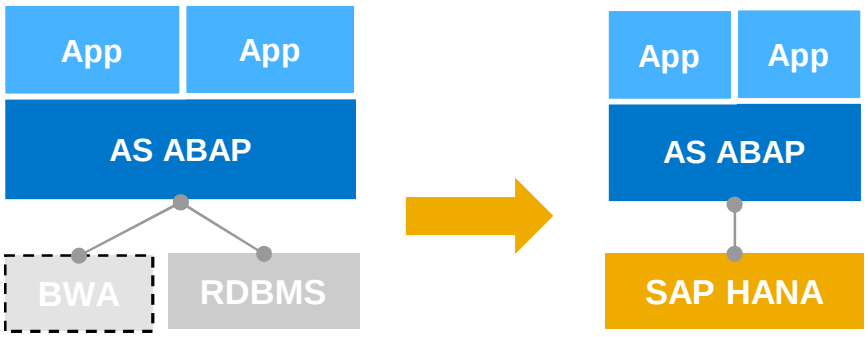
ABAP Platform and SAP HANA

Scenarios and transition options

Side-by-Side



Primary Persistence



Influencing factors for Side-by-Side vs. Primary Persistence

- Innovation speed (e.g. prerequisities, limitations)
- Stability (e.g. impact on productive systems)
- TCO (e.g. landscape complexity, data integration)

Transition options for primary persistence

- New installation (chance to „clean up“)
- Copy -> Upgrade -> Migrate („before/after“ with fallback)
- In-place migration (keep landscape (e.g. SID and server))

Technology aspects of ABAP on SAP HANA

Key technology aspects of SAP HANA

- Support for **multi-core** architecture (→benefit from massive parallelization)
- Row and **column based** data store (→very fast aggregation and search)
- High data **compression** (→ make use of real-life / sparse fill of tables)

Key consequences for developing ABAP on SAP HANA

- **Database access** becomes center of attention (→ good DB / SQL knowledge is key)
- **Performance** is not only an expert domain anymore (→ tools and guidelines)

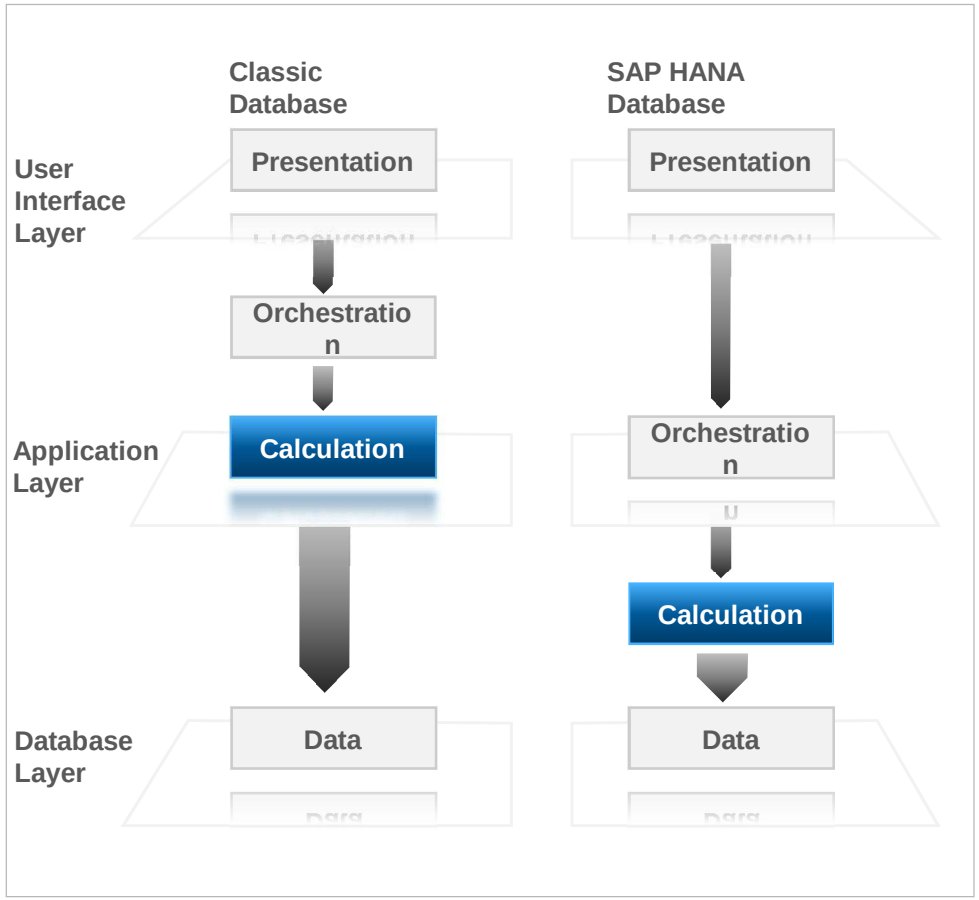
The consequence: A paradigm shift
“Code Pushdown“ or “Code-2-Data“

Characteristics

- Data processing code is running inside HANA
- Less data transfer between HANA and ABAP
- Reuse possible in non-ABAP scenarios

Affected development domains include

- Data modelling & access
- Process and display logic
- Authorization checks

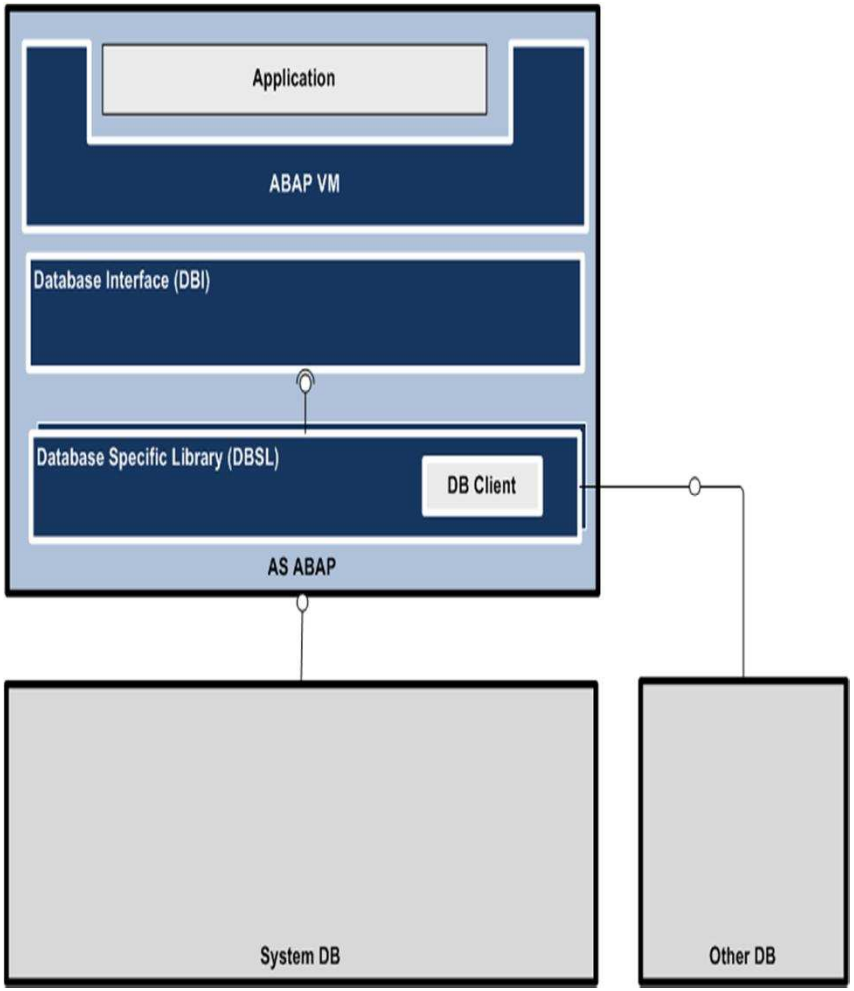


ABAP database architecture in a nutshell

- **Database integration: DBI and DBSL**
- The DBI provides a **database independent** interface and additional services like automatic client handling and the ABAP table buffer.
- The DBSL connects to the respective database.

- **Database Users and Schemas**
- The ABAP system runs with one DB user („SAP<SID>" or "SAPR3"). This user has many priviledges.
- The ABAP system stores all data in the database schema corresponding to this user.

- **Multiple DB clients**
- The server can have multiple DB clients installed allowing to connect to different remote databases.



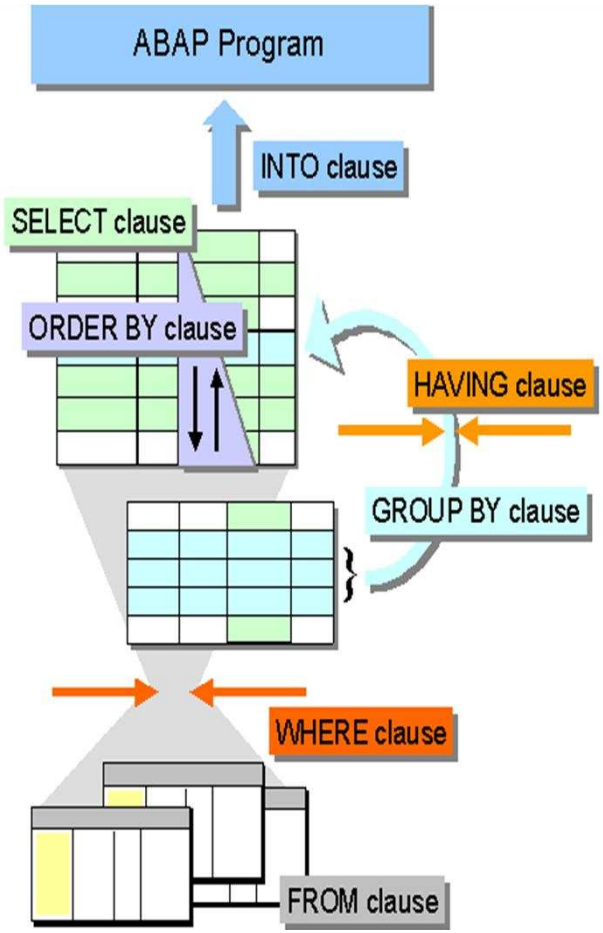
Open SQL in ABAP

Open SQL in a nutshell

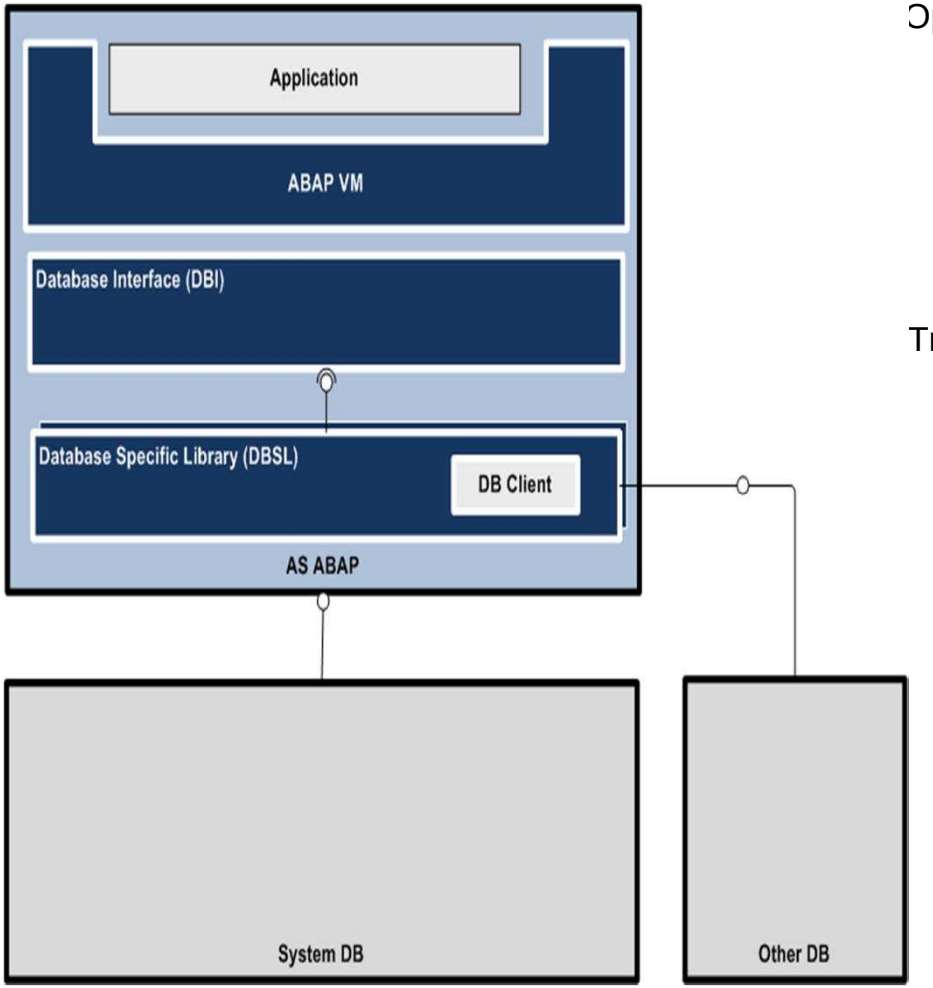
- Open SQL provides a uniform syntax and semantics for all of the database systems supported by SAP NetWeaver.
- ABAP programs that only use Open SQL statements will work in any SAP system, regardless of the database system in use.
- Open SQL statements can only work with database tables that have been created in the ABAP Dictionary.
- Open SQL can be used via secondary database connections

Relation of Open SQL to the DML/DDL/DCL aspects of SQL

- Open SQL covers the DML aspects
- The ABAP dictionary tools control the DDL aspects
- The DCL aspects are not reflected in standard ABAP; instead data access control is managed by the ABAP authorization concept.



Example: Translation of Open SQL to native SQL



Open SQL statement

```
SELECT carrid connid fldate
       FROM sflight
       INTO CORRESPONDING FIELDS OF TABLE sflight_tab
       FOR ALL ENTRIES IN entry_tab
       WHERE carrid = entry_tab-carrid AND
             connid = entry_tab-connid.
```

Translated to native SQL (taken from SQL trace; ST05)

```
SQL Statement

SELECT
  "CARRID" , "CONNID" , "FLDATE"
FROM
  "SFLIGHT"
WHERE
  ( "MANDT" = ? AND "CARRID" = ? AND "CONNID" = ? ) OR ( "MANDT" = ? AND
  "CARRID" = ? AND "CONNID" = ? ) OR ( "MANDT" = ? AND "CARRID" = ? AND
  "CONNID" = ? ) OR ( "MANDT" = ? AND "CARRID" = ? AND "CONNID" = ? ) OR (
  "MANDT" = ? AND "CARRID" = ? AND "CONNID" = ? )

Variable

A0(CH,3)  = 005
A1(CH,3)  = LH
A2(NU,4)  = 2402
A3(CH,3)  = 005
A4(CH,3)  = LH
A5(NU,4)  = 0402
A6(CH,3)  = 005
A7(CH,3)  = LH
A8(NU,4)  = 0400
A9(CH,3)  = 005
A10(CH,3) = UA
A11(NU,4) = 0941
A12(CH,3) = 005
A13(CH,3) = JL
A14(NU,4) = 0408
```

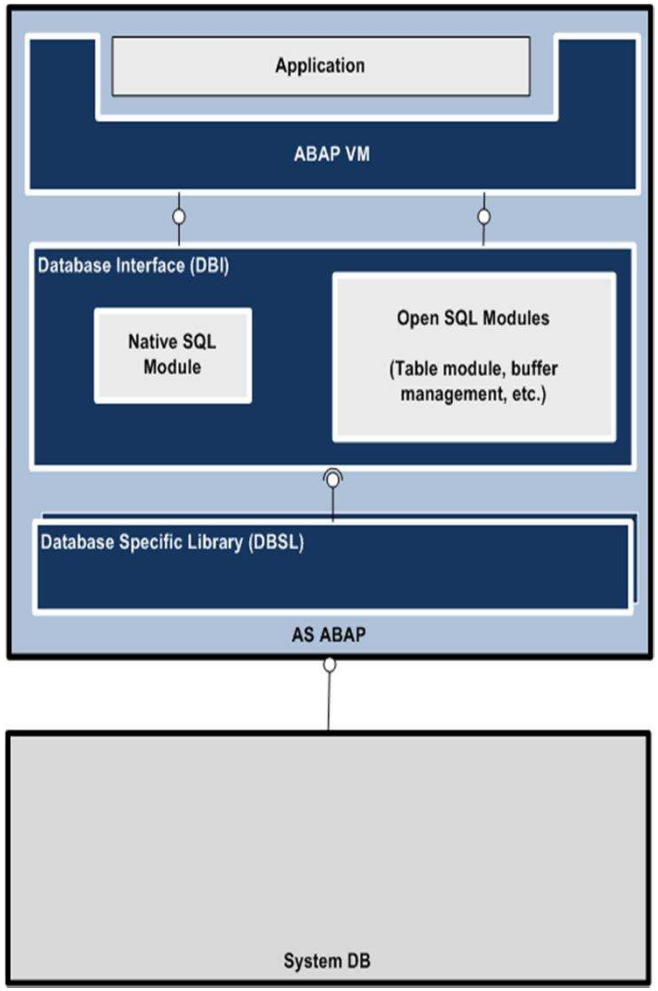
Accessing the database directly via native SQL

Native SQL in a nutshell

- Native SQL is only loosely integrated into ABAP, but allows access to all of the functions contained in the programming interface of the respective database system.
- Native SQL statements are not checked for correct syntax, but instead are sent directly to the database system.
- All tables and views in all schemas can be accessed (if the corresponding database user has sufficient privileges).
- There is no automatic client handling, nor table buffering.

EXEC SQL vs. ADBC

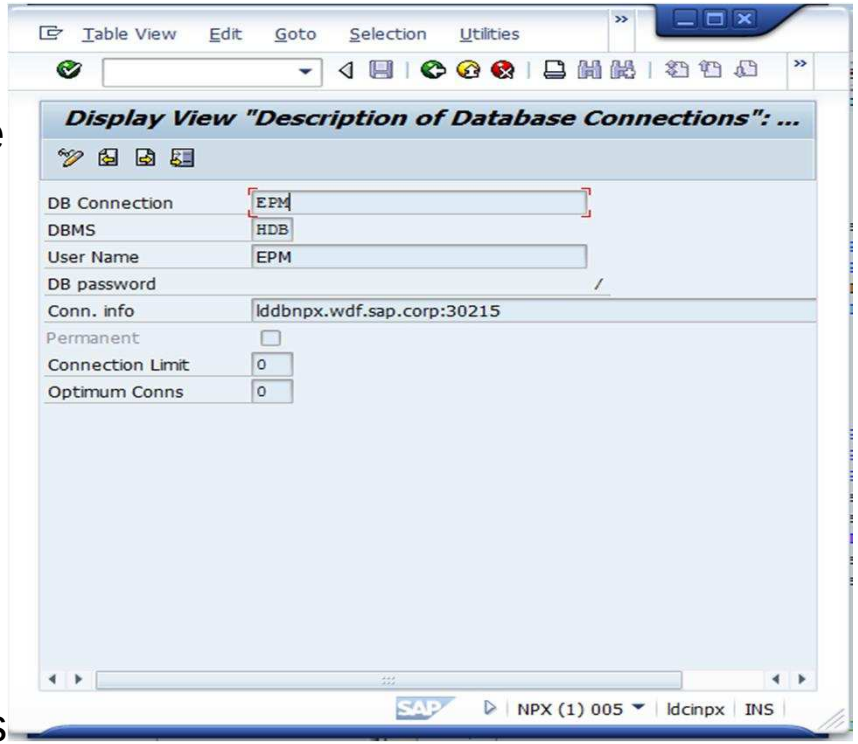
- Native SQL can be used via **EXEC SQL** (and related) statements or the ABAP Objects based **ADBC** (ABAP Database Connectivity) API.
- The general recommendation is to **prefer ADBC** because of better flexibility (e.g. flexible package size) and object orientation.



Secondary database connections

Secondary connections

- can be used to access local or remote database systems
- are maintained via SM30 for table DBCON; entries can be transported
- require specification of connection data including user (=DB schema) and password
- are supported in the Open SQL syntax by using the CONNECTION supplement
- form an own transaction context



Service note 1597627 describes the prerequisites and procedure for setting up a secondary connection to HANA.

ABAP Database Connectivity(ADBC)

Have a look at the Classbuilder (SE24) or Transaction SE80!

CL_SQL_CONNECTION

- **GET_CONNECTION**
- **CREATE_STATEMENT** and **PREPARE_STATEMENT**
- **ROLLBACK** and **COMMIT**

CL_SQL_PREPARED_STATEMENT / CL_SQL_STATEMENT

- **PREPARE / CLOSE** – Prepare / release an SQL Statement
- **SET_PARAM** - Set an Input/Output Parameter (variants for CLOB, BLOB, STRUCT, TABLE (available soon))
- **PREPARED_QUERY, PREPARED_UPDATE** - Execute a Prepared Query / DML Operation
- **EXECUTE_DDL, EXECUTE_QUERY, EXECUTE_UPDATE** - Execute DDL, Query, DML (Insert, Update, Delete)

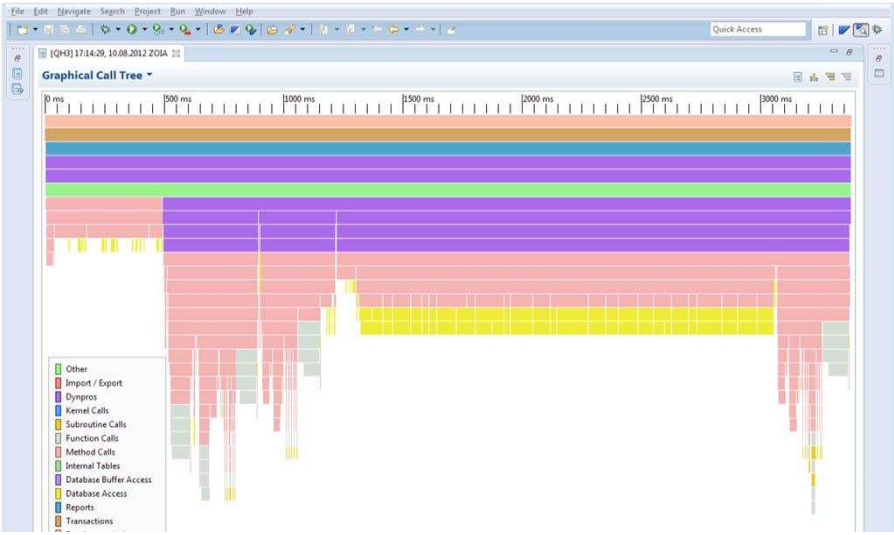
CL_SQL_RESULT_SET

- **SET_PARAM** - Set an Input/Output Parameter (variants for CLOB, BLOB, STRUCT, TABLE)
- **NEXT, NEXT_PACKAGE** – Read next record in the resulting set, or next set of records for internal tables

How to detect optimizing potential on SAP HANA?
Performance tools in the Application Server ABAP

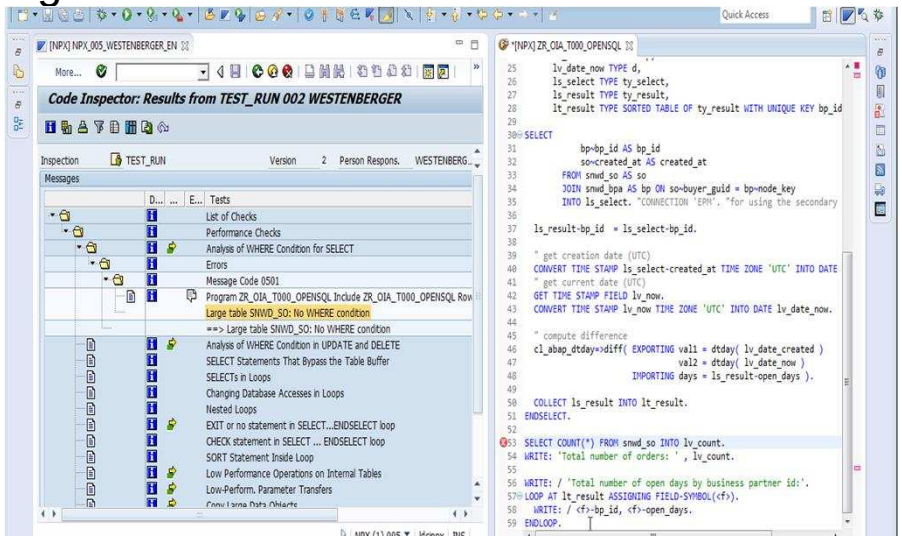
Tools for runtime analysis

- New ABAP profiler in Eclipse based on SAT* (enriched with graphical representations)
- Proven SQL Trace, STAD, DBA Cockpit



Static code checks and guidelines

- Detect certain anti-patterns in DB access (reported with priority based on table size, etc.)
- Integrated improvement proposals and guidelines

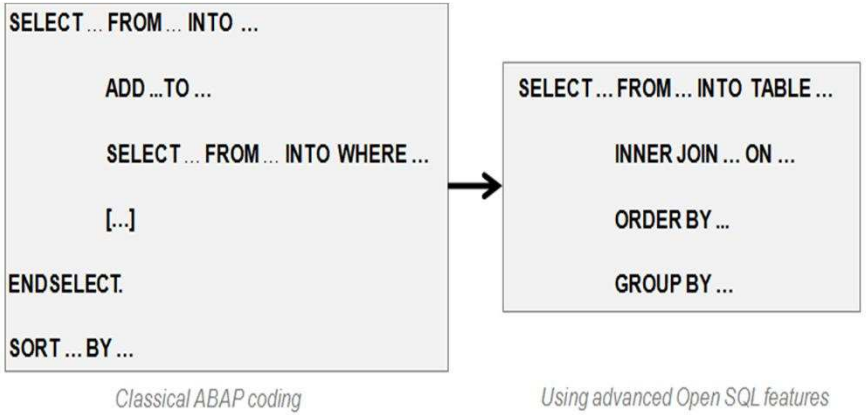


* SAT = Single Activity Trace (Runtime Analysis tool)

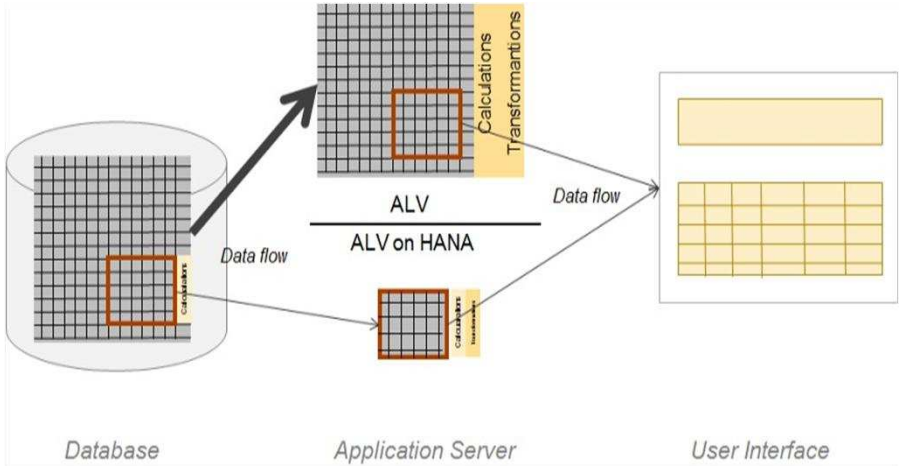
How to optimize existing ABAP code for SAP HANA

Two concrete examples

- **Use the power of Open SQL**
- Use sorting, aggregations, joins, sub-selects, etc.
- Reduce database roundtrips and transfering too much data into application server
- Allows implicitly to benefit from parallelization on SAP HANA



- **Leverage ALV optimized for SAP HANA**
- Option to describe data declaratively instead of passing large internal tables
- Optimized HANA database access based on user interface needs
- Usable in SAP GUI and Web Dynpro / Floorplan Manager

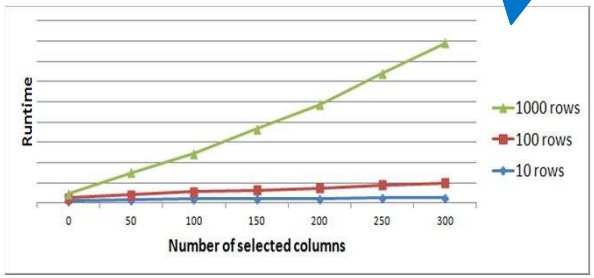


Some concrete best practices for optimization

Qualitative only

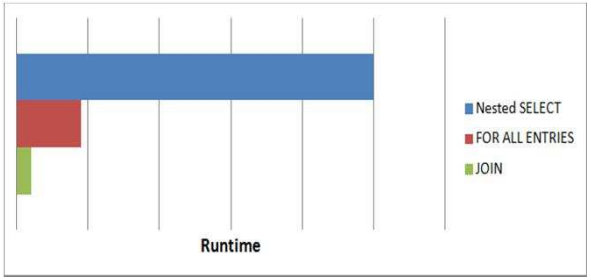
Field list optimization
SELECT … FROM … WHERE … UP TO n ROWS

The more rows are selected, the more important becomes the optimization for field lists. Large factors (>20) are possible for 1000 rows.



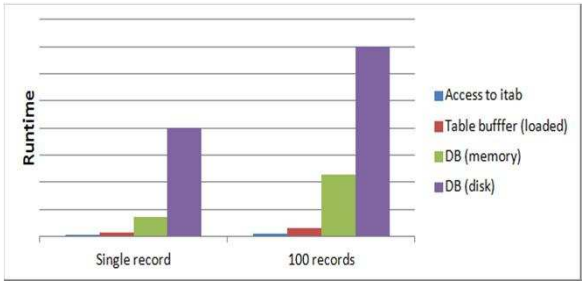
Usage of joins instead of nested SELECT statements (or FOR ALL ENTRIES)

Proper usage of JOINs becomes more important on HANA due to column storage. General rule: runtime for
JOIN << FOR ALL ENTRIES << Nested SELECT



Usage of ABAP table buffer according to existing guidelines

Basic rules still apply in general:
Access times in ABAP coding:
Internal table << table buffer << DB cache / HANA << standard DB disk



…. …. ….

More best practices and guidelines can be found at: http://scn.sap.com/community/abap-for-hana

ABAP Development Tools for SAP NetWeaver

SAP's new ABAP IDE built on Eclipse™

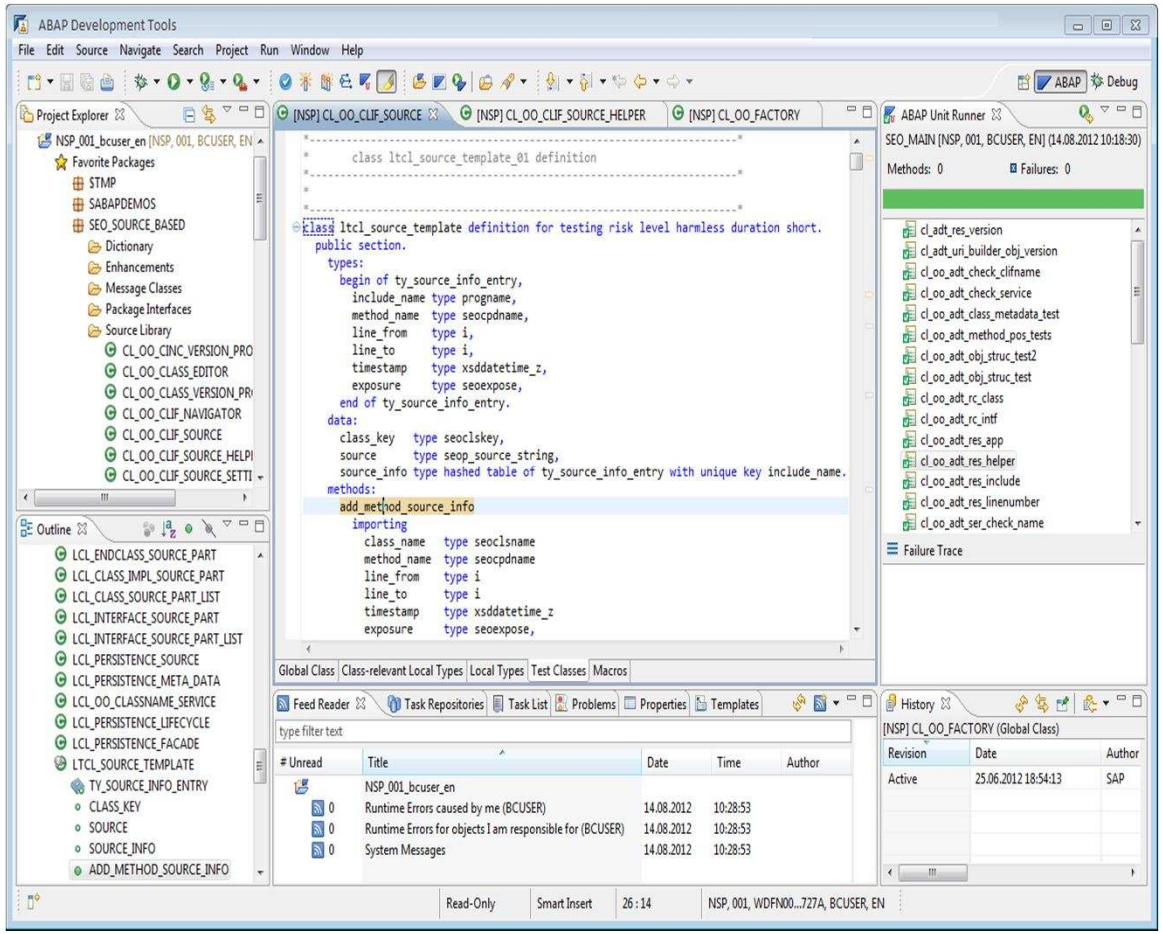
The ABAP Development Tools integrate tightly with all Eclipse-based development tools of SAP's strategic product areas cloud, mobility and in-memory providing a highly productive E2E development environment.

Highlights

- Evolution of the ABAP workbench built on Eclipse offering excellent user experience and assistance
- One IDE for all development tasks: SAP HANA modeling, ABAP development, HTML5 UI, ...
- Powerful search and navigation, advanced source code editing and refactoring capabilities
- Built-in extensibility: ADT SDK (lab preview)

More Information

- SCN: http://scn.sap.com/community/abap/eclipse
- Trial: http://scn.sap.com/docs/DOC-29607
- Youtube: http://youtu.be/BXg7xXrEAUw



Integrated development options across ABAP and HANA
Consuming HANA views in ABAP

SAP HANA offers advanced **view modeling,** e.g.

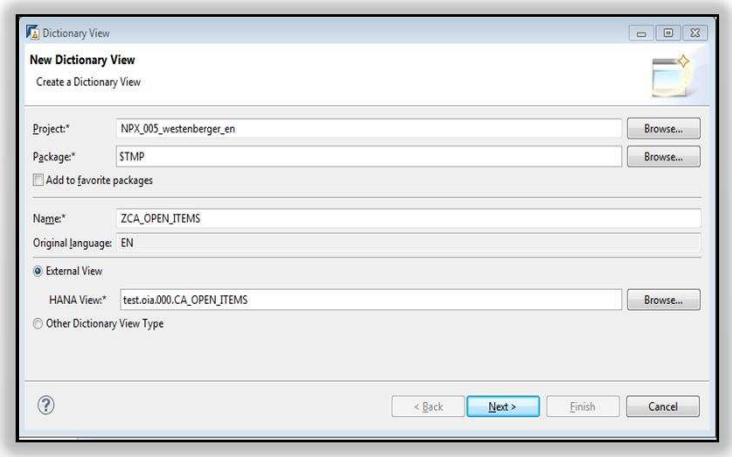
- Attribute views (join views)
- Analytic views (star schemas)
- Calculation views (modeled or coded via SQL script)

With ABAP < 7.40 these views can be accessed low-level via ADBC.

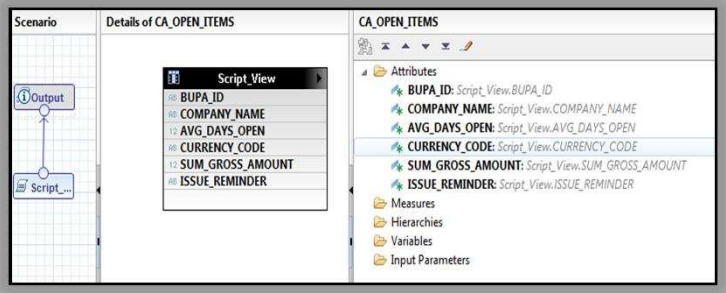
With ABAP 7.40 they are natively supported in ABAP

- Access possible via standard Open SQL
- Support for automatic client handling
- Mapping to DDIC types possible

```
select * from zca_open_items into table lt_result
        where avg_days_open > 90 and sum_gross_amount >= 100000.
```



Consumption from ABAP



Calculation view in SAP HANA

Integrated development options across ABAP and HANA

Calling HANA database procedures from ABAP

SAP HANA offers writing **stored procedures** in SQL Script – a extension to SQL - for expressing data intensive application logic**.**

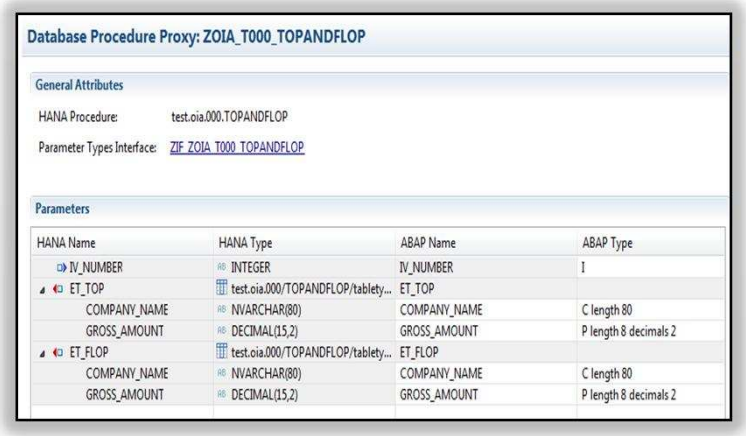
With ABAP < 7.40 stored procedures can be called using ADBC, which requires

- Manual handling of transfer tables for input and output parameters via temporary tables or result views
- Manual mapping of database types to DDIC types

With ABAP 7.40 they are natively supported in ABAP

- Exporting/Importing parameters like for function modules (including mapping parameter to DDIC types)

```
CALL DATABASE PROCEDURE zoia_t000_topandflop( EXPORTING  iv_number = lv_number
                                              IMPORTING  et_top    = lt_top
                                                         et_flop   = lt_flop ).
```



Database Procedure Proxy: ZOIA_T000_TOPANDFLOP

General Attributes

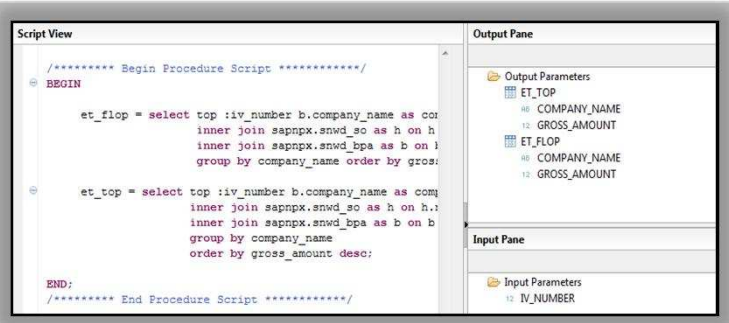
HANA Procedure: test.oia.000.TOPANDFLOP

Parameter Types Interface: ZIF_ZOIA_T000_TOPANDFLOP

Parameters

HANA Name	HANA Type	ABAP Name	ABAP Type
IV_NUMBER	INTEGER	IV_NUMBER	I
ET_TOP	test.oia.000/TOPANDFLOP/tablety...	ET_TOP	
COMPANY_NAME	NVARCHAR(80)	COMPANY_NAME	C length 80
GROSS_AMOUNT	DECIMAL(15,2)	GROSS_AMOUNT	P length 8 decimals 2
ET_FLOP	test.oia.000/TOPANDFLOP/tablety...	ET_FLOP	
COMPANY_NAME	NVARCHAR(80)	COMPANY_NAME	C length 80
GROSS_AMOUNT	DECIMAL(15,2)	GROSS_AMOUNT	P length 8 decimals 2

Invocation from ABAP



Stored procedure in SAP HANA

Demo

THANK YOU FOR PARTICIPATING

Please provide feedback on this session by completing a short survey via the event mobile application.

SESSION CODE: 0702

For ongoing education on this area of focus, visit www.ASUG.com

© 2013 SAP AG. All rights reserved.