# SAP HANA

Lesson Name: HANA Specific Code-to-Data

## Lesson Objectives

After completing this lesson, participants will be able to -

- Know about native SQL using SAP HANA
  - Viewing Tables
  - Select and From
  - Where Clauses
  - Functions
  - Group By
  - Order By
  - Having
  - Top
  - Create
  - Insert
  - Update
  - Delete
  - Joins

## Lesson Objectives

  - Sub Selects
  - Unions
  - Drop
  - Views
  - Schemas
  - Table Types
- Understand ADBC
- Use ADBC to execute Native SQL statements
- AMDP
- AMDP Debugging

## Contents

# Native SQL using SAP HANA

VIEWING TABLES

- Syntax

  Select column1, column2 from ( select column1,column2 from “table_name”)

- example of code is:

```
SELECT TOP 200 “CUSTOMERID” ,“CITYID”, “COUNTRYID”, “REGIONID”,
“COMPANYNAME”, “POSTALCODE”, “CITYNAME”, “COUNTRYNAME”,
“REGIONNAME” FROM ( SELECT * FROM “STS” . “DIMCUSTOMER” ) TMP
```

## Native SQL using SAP HANA

SELECT AND FROM

- Syntax

  Select COLUMN1, COLUMN2 FROM “TABLE_NAME”

- example of code is:

```
SELECT * FROM “STSFLAT”.“STSCUSTOMERFLATFILE”

SELECT  COMPANYNAME  FROM  STS.DIMCUSTOMER
```

# Native SQL using SAP HANA

WHERE

- Syntax

  SELECT COLUMN1 COLUMN 2 FROM "TABLE1"
  where CUSTOMID ='2'

- example of code is:

SELECT * FROM "STS"."DIMCUSTOMER" where CITYNAME = 'LONDON'

SELECT * FROM "STS"."DIMCUSTOMER" where CITYNAME like 'R%'

## Native SQL using SAP HANA

FUNCTIONS

- Syntax

  SELECT COUNT(10) FROM "TABLE1' where CUSTOMID='2'

- example of code is:

SELECT  COUNT(*)  FROM "STS"."DIMCUSTOMER" where COUNTRYNAME = 'USA'

SELECT  COUNT(QUANTITY)  FROM "STS"."DIMCUSTOMER" where COUNTRYNAME = 'USA'

## Native SQL using SAP HANA

GROUP BY

- Syntax

  select column1 as field_name from table_name GROUP BY column1

- example of code is:

```
SELECT  countryname as COUNTRY,
SUM(netsales) as TOTAL_SALES
FROM “STSFLAT”.“STSCUSTOMERFLATFILE”
GROUP BY countryname
HAVING sum(netsales) > 4000000
order by COUNTRYNAME
```

## Native SQL using SAP HANA

ORDER BY

- Syntax

  SELECT COLUMN1 COLUM2 FROM “TABLE1” order by VARIABLE

- example of code is:

```
SELECT  * FROM “STS”.“DIMCUSTOMER” order by CUSTOMERID
```

## Native SQL using SAP HANA

HAVING

- Syntax
  select column1 as field_name from table_name HAVING condition

- example of code is:

```
SELECT  countryname as COUNTRY,
SUM(netsales) as TOTAL_SALES
FROM “STSFLAT”.“STSCUSTOMERFLATFILE”
GROUP BY countryname
HAVING sum(netsales) > 4000000
order by COUNTRYNAME
```

## Native SQL using SAP HANA

TOP

- Syntax

  select TOP no_var column1 as field_name from table_name

- example of code is:

```
SELECT  TOP 5 countryname as COUNTRY,
SUM(netsales) as TOTAL_SALES
FROM “STSFLAT”.“STSCUSTOMERFLATFILE”
GROUP BY countryname
HAVING sum(netsales) > 4000000
order by COUNTRYNAME
```

# Native SQL using SAP HANA

CREATE

- Syntax

```
CREATE COLUMN1 TABLE1 "VARIABLE1"."VARIABLE2"
{SUPPLIERID" INTEGER CS_INT NOT NULL ,
    "CITYID" INTEGER CS_INT,
  PRIMARY KEY ("SUPPLIERID")}
```

- example of code is:

```
CREATE COLUMN TABLE "XTRA"."DIMCUSTOMERV2"
 {SUPPLIERID" INTEGER CS_INT NOT NULL ,
"CITYID" INTEGER CS_INT,
"COUNTRYID" INTEGER CS_INT,
"COMPANYNAME" NVARCHAR (20),
PRIMARY KEY ("SUPPLIERID")}
```

## Native SQL using SAP HANA

INSERT

- Syntax

  insert into table_name values { variable, field_name}

- example of code is:

```
insert into “XTRA” . “MYTESTTABLE” values
{
2,12345, ‘VW PASSAT’
};
```

## Native SQL using SAP HANA

UPDATE

- Syntax

  update table_name set field_name = variable where condition

- example of code is:

```
update “XTRA” . “MYTESTTABLE”
set CARREGISTRATION = 12345
where CARID = 2
```

## Native SQL using SAP HANA

DELETE

- Syntax

  delete from table_name where condition

- example of code is:

```
delete from “XTRA” . “MYTESTTABLE”
where CARID = 2
```

# Native SQL using SAP HANA

JOIN

- Syntax

  select column1 from table_name1 inner join table_name2 on condition.

- example of code is:

```
select
T0. “COMPANYNAME”,
T1. “NETSALES”
from
“STS”. “DIMCUSTOMER” T0 inner join “STS”.”FCTCUSTOMERORDER” T1
on T0. “CUSTOMERID” = T1. “CUSTOMERID”
```

## Native SQL using SAP HANA

SUB SELECT

- Syntax

```
select column1 from table_name where (condition)
having (condition1)
(
select (condition) from table_name1
)
```

## Native SQL using SAP HANA

SUB SELECT

- example of code is:

```
SELECT COMPANYNAME AS COMPANY, ROUND(AVG (NETSALES),0) AS
AVERAGE_SALES
FROM “STS”. “DIMSUSTOMER”, “STS”. “FCTCUSTOMERORDER”
WHERE “STS”. “DIMCUSTOMER” = “STS”.”FCTCUSTOMERORDER”
GROUP BY CMPANYNAME
HAVING AVG (NETSALES) >
(
SELECT ROUND (AVG (NETSALES), 2) as AVERAGE
FROM “STS”. “FCTCUSTOMERORDER”
)
ORDER BY COMPANYNAME
```

## Native SQL using SAP HANA

UNION

- Syntax

  select column1 from table1 UNION select column2 from  table2

- example of code is:

```
{ SELECT COMPANYNAME AS COMPANY FROM “STS” . “DIMCUSTOMER”}
UNION
{ SELECT COMPANYNAME AS COMPANY FROM “XTRA” .
“ADDITIONALPROSPECTS”}
```

# Native SQL using SAP HANA

## DROP

- Syntax

  drop table table_name

- example of code is:

```
drop table STS.AAA ;

create table STS.AAA as
{
select *  from “STS”.”DIMCUSTOMER”
};
```

## Native SQL using SAP HANA

VIEW

- Syntax

  create view view_name as

  select column1 column2 as field_name from table_name

- example of code is:

```
create view STS.STS_VIEW as
SELECT  countryname as COUNTRY,
SUM(netsales) as TOTAL_SALES
FROM “STSFLAT”.“STSCUSTOMERFLATFILE”
GROUP BY countryname
HAVING sum(netsales) > 4000000
order by COUNTRYNAME
```

## Native SQL using SAP HANA

SCHEMA

- Syntax

  create / drop schema schema_name

- example of code is:

```
create schema newuseradditionalschema owned by newuser

drop schema “NEWUSERADDITIONALSCHEMA”
```

# Native SQL using SAP HANA

## TABLE TYPES

- Syntax

  alter table table_name alter type row or column

- example of code is:

```
create column table sts.columnstoretable
(columna int)

alter table “STS”.”COLUMNSTORETABLE” alter type row;
```

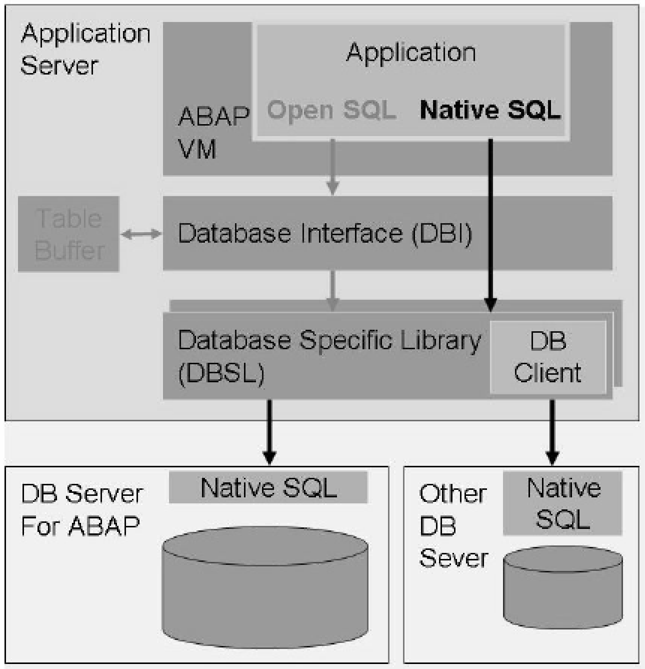
# ADBC

## Business Example

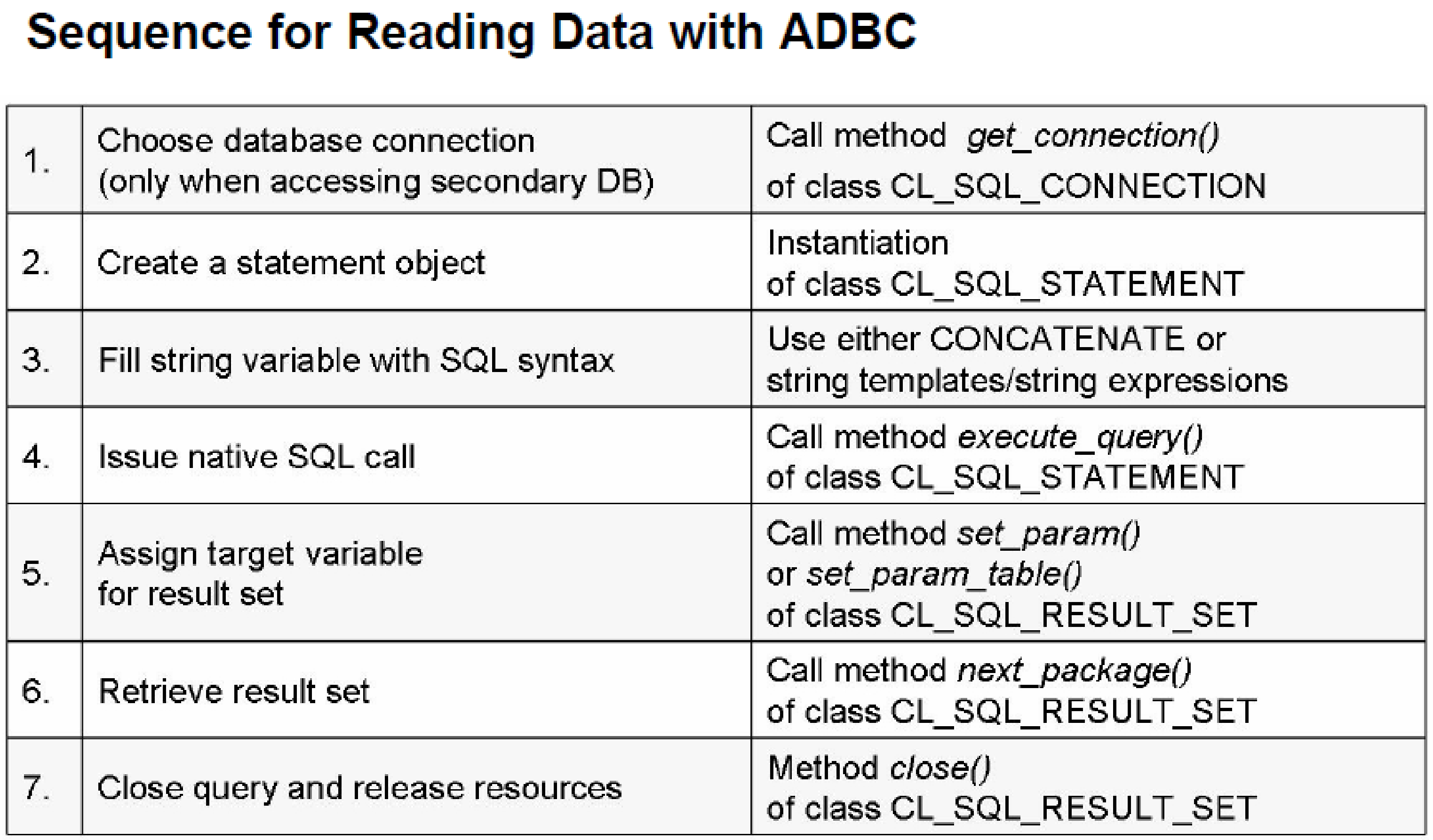
Your company has recognized SAP HANA as an important strategic topic and has asked you to refresh your ABAP skills, paying particular attention to those required to develop or review code that leverages the strengths of SAP HANA.

### ADBC

- ***ABAP Database Connectivity*** *(ADBC)* is an Object-based API
- ADBC allows native SQL access providing:
  - Flexibility
  - Where used list
  - Error handling
- Main classes are:
  - CL_SQL_CONNECTION
  - CL_SQL_STATEMENT
  - CL_SQL_RESULT_SET



# ADBC

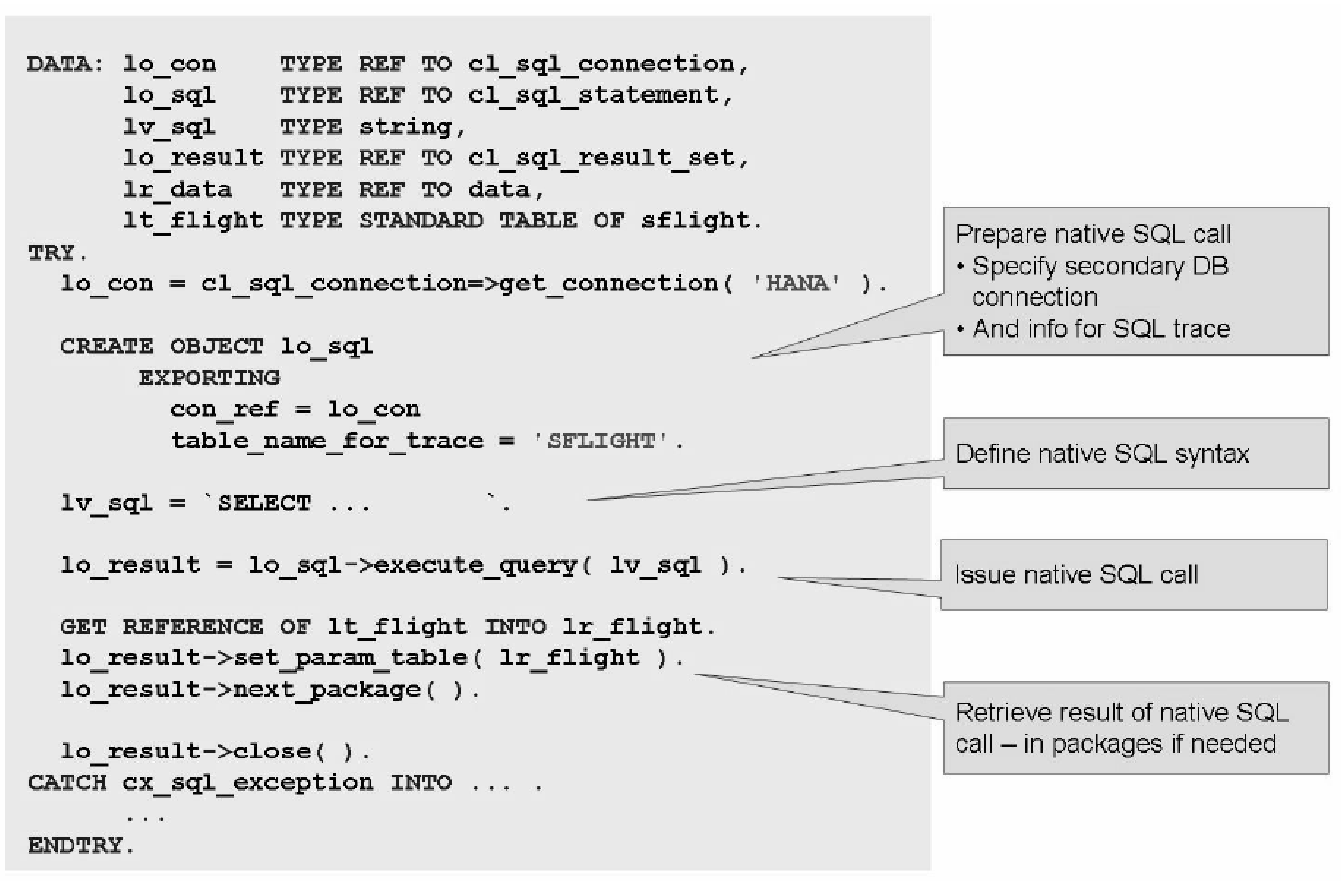


## Sequence for Reading Data with ADBC

| | | |
|---|---|---|
| 1. | Choose database connection (only when accessing secondary DB) | Call method *get_connection()* of class CL_SQL_CONNECTION |
| 2. | Create a statement object | Instantiation of class CL_SQL_STATEMENT |
| 3. | Fill string variable with SQL syntax | Use either CONCATENATE or string templates/string expressions |
| 4. | Issue native SQL call | Call method *execute_query()* of class CL_SQL_STATEMENT |
| 5. | Assign target variable for result set | Call method *set_param()* or *set_param_table()* of class CL_SQL_RESULT_SET |
| 6. | Retrieve result set | Call method *next_package()* of class CL_SQL_RESULT_SET |
| 7. | Close query and release resources | Method *close()* of class CL_SQL_RESULT_SET |

# ADBC

## Coding Example: ABAP Database Connectivity (ADBC)



```
DATA: lo_con    TYPE REF TO cl_sql_connection,
      lo_sql    TYPE REF TO cl_sql_statement,
      lv_sql    TYPE string,
      lo_result TYPE REF TO cl_sql_result_set,
      lr_data   TYPE REF TO data,
      lt_flight TYPE STANDARD TABLE OF sflight.
TRY.
  lo_con = cl_sql_connection=>get_connection( 'HANA' ).

  CREATE OBJECT lo_sql
       EXPORTING
         con_ref = lo_con
         table_name_for_trace = 'SFLIGHT'.

  lv_sql = `SELECT ...       `.

  lo_result = lo_sql->execute_query( lv_sql ).

  GET REFERENCE OF lt_flight INTO lr_flight.
  lo_result->set_param_table( lr_flight ).
  lo_result->next_package( ).

  lo_result->close( ).
CATCH cx_sql_exception INTO ... .
      ...
ENDTRY.
```

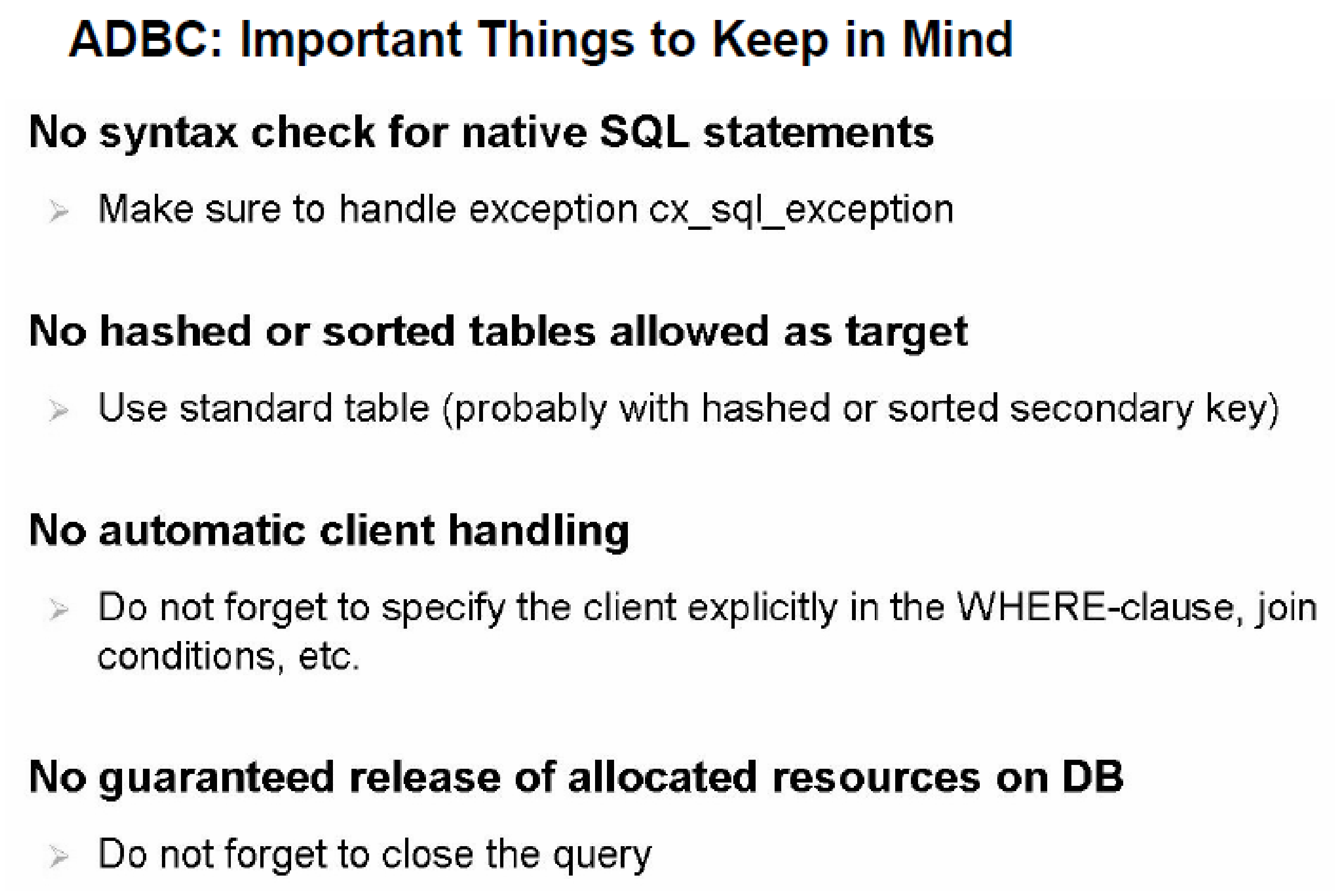
Prepare native SQL call
- Specify secondary DB connection
- And info for SQL trace

Define native SQL syntax

Issue native SQL call

Retrieve result of native SQL call – in packages if needed

## ADBC



### ADBC: Important Things to Keep in Mind

**No syntax check for native SQL statements**

- Make sure to handle exception cx_sql_exception

**No hashed or sorted tables allowed as target**

- Use standard table (probably with hashed or sorted secondary key)

**No automatic client handling**

- Do not forget to specify the client explicitly in the WHERE-clause, join conditions, etc.

**No guaranteed release of allocated resources on DB**

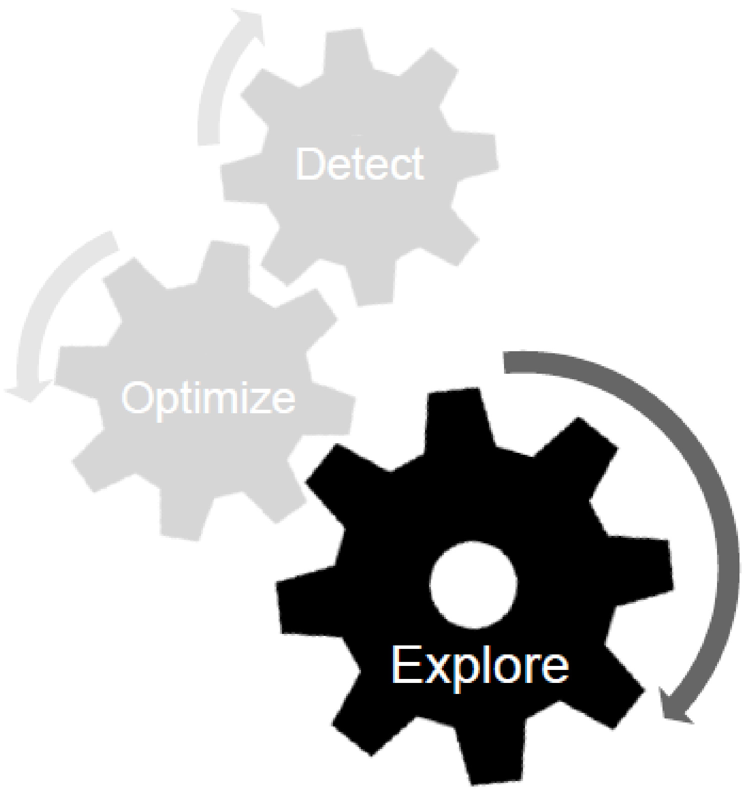
- Do not forget to close the query

# AMDP

## ABAP Managed Database Procedures

Migration to SAP HANA

1. **Detect**
   - Functional correctness
   - Performance optimization potential
2. **Optimize**
   - Database-oriented programming
3. **Explore**
   - Use SAP HANA-specific features
   - Rethink & innovate

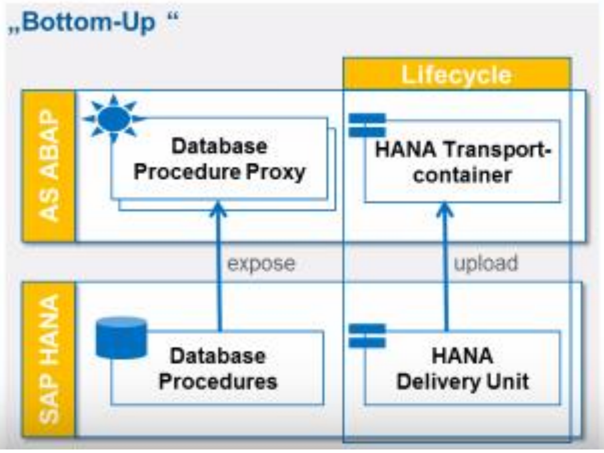


For AMDP, we are step 3 Explore.

# AMDP

## Current way for using HANA Procedures in ABAP

“Bottom-Up” Approach with Database Procedure Proxies



**ABAP and HANA servers involved** for developing, managing and calling HANA procedures in ABAP

**HANA procedure developed in the HANA Studio**

**Database Procedure Proxy created in the ABAP Development Tools**

**HANA Transport Container and HANA Delivery Unit required**

- For integrating the HANA content in the standard ABAP transport mechanism

# AMDP

## New way of using HANA Database Procedures in ABAP
ABAP Managed Database Procedure (AMDP)

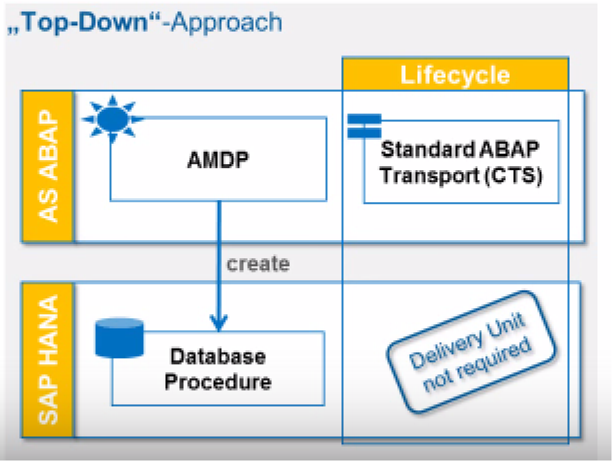
**ABAP server as sole *Master* for developing, managing and calling database procedures**

**AMDP provided as methods of global classes marked with tag interfaces (aka AMDP class)**

**HANA procedure created at the first call of the AMDP method**

**Transport only required for the AMDP class**

**Only ABAP Development Tools required**



# AMDP

## Code-to-Data Paradigm

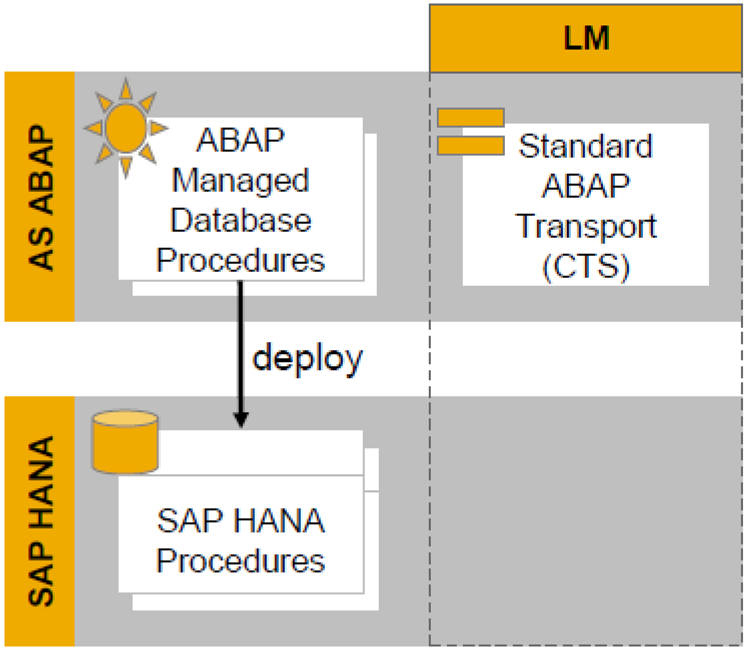
- Supported through embedding native database procedure coding

## 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


LM
AS ABAP
ABAP Managed Database Procedures
Standard ABAP Transport (CTS)
deploy
SAP HANA
SAP HANA Procedures

# AMDP

## 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.
```

# AMDP

**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.
```

# AMDP

**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)
      ).
```

## AMDP

**Catchable Exceptions**

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

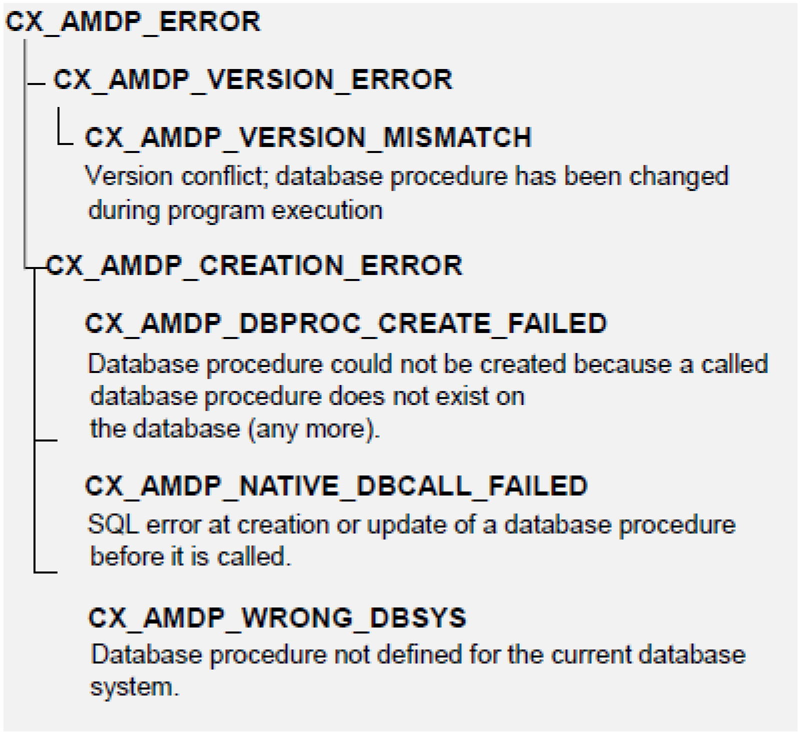
  <ERROR_NAME> → CX_<ERROR_NAME>

- To-Dos for AMDP Developers/Consumers:
  - Add RAISING clause to the AMDP method definition
  - Enclose the AMDP call in a TRY… CATCH block

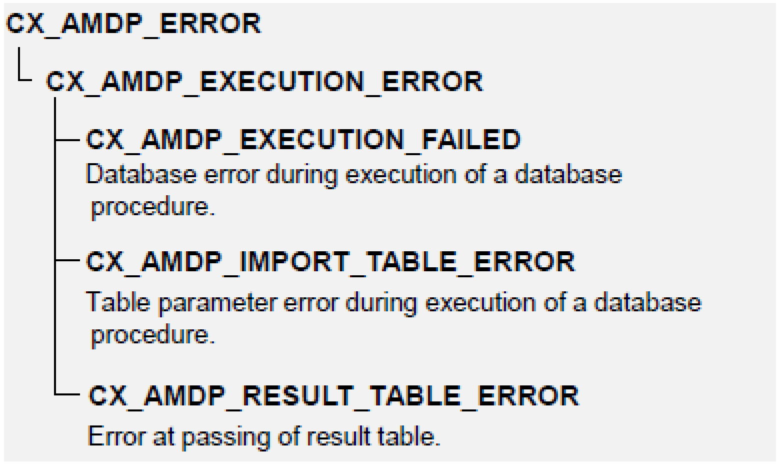
```
"definition
METHODS <method_name>
        <method_interface>
      RAISING cx_amdp_error.
...

"consumption
TRY.
    <method_call>

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

# AMDP

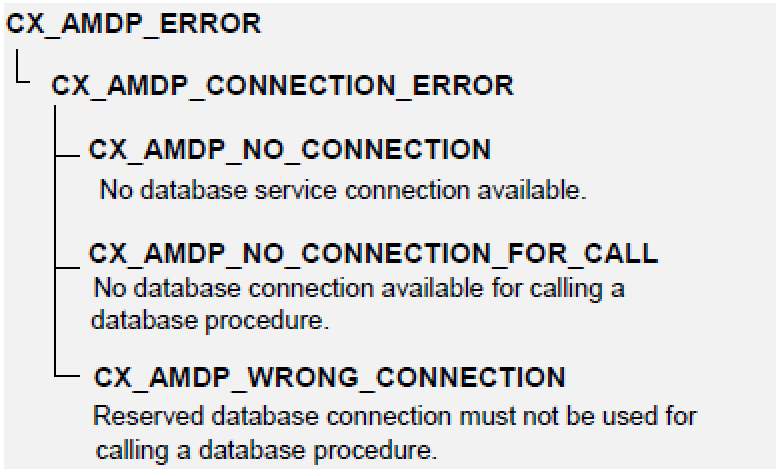
- Structure of AMDP Exception Classes



- CX_AMDP_ERROR
  - CX_AMDP_VERSION_ERROR
    - CX_AMDP_VERSION_MISMATCH
      Version conflict; database procedure has been changed during program execution
  - CX_AMDP_CREATION_ERROR
    - CX_AMDP_DBPROC_CREATE_FAILED
      Database procedure could not be created because a called database procedure does not exist on the database (any more).
    - CX_AMDP_NATIVE_DBCALL_FAILED
      SQL error at creation or update of a database procedure before it is called.
    - CX_AMDP_WRONG_DBSYS
      Database procedure not defined for the current database system.



- CX_AMDP_ERROR
  - CX_AMDP_EXECUTION_ERROR
    - CX_AMDP_EXECUTION_FAILED
      Database error during execution of a database procedure.
    - CX_AMDP_IMPORT_TABLE_ERROR
      Table parameter error during execution of a database procedure.
    - CX_AMDP_RESULT_TABLE_ERROR
      Error at passing of result table.



- CX_AMDP_ERROR
  - CX_AMDP_CONNECTION_ERROR
    - CX_AMDP_NO_CONNECTION
      No database service connection available.
    - CX_AMDP_NO_CONNECTION_FOR_CALL
      No database connection available for calling a database procedure.
    - CX_AMDP_WRONG_CONNECTION
      Reserved database connection must not be used for calling a database procedure.

## AMDP

DEMO

## AMDP Debugging

For AMDP Debugging, we need 3 users.

ABAP Developer User ,
Who will run the report program

HANA User of same system

Debug user is a HANA User who will set the debugging

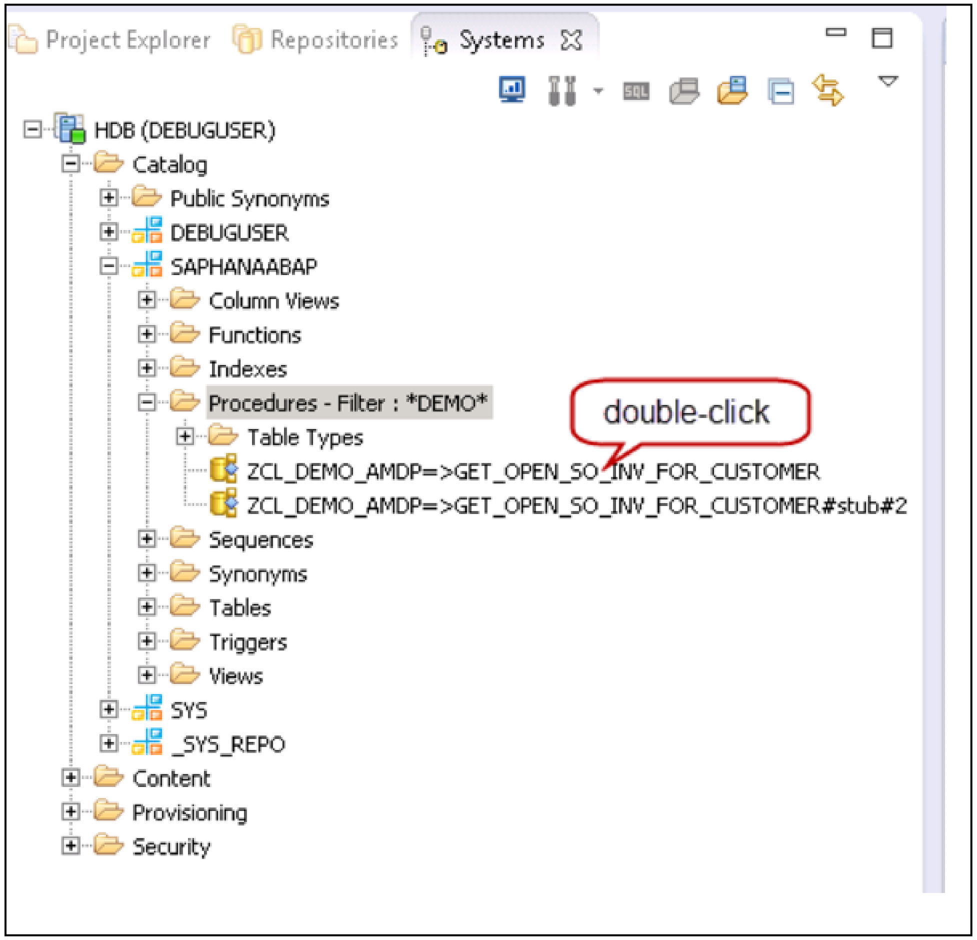
## AMDP Debugging

- Set a break point by Debug User in Catalog DB Procedure
- Create debug configuration with Filter criteria
- Start SQL script debugger ( Attach session of HANA User )
- Developer now execute the ABAP report and AMDP called

## AMDP Debugging

- Set a break point by Debug User in Catalog DB Procedure
- Go to HANA Development perspective with Debug user.
- Open SAP developer User schema
- Open the respective AMDP HANA Procedure

# AMDP Debugging

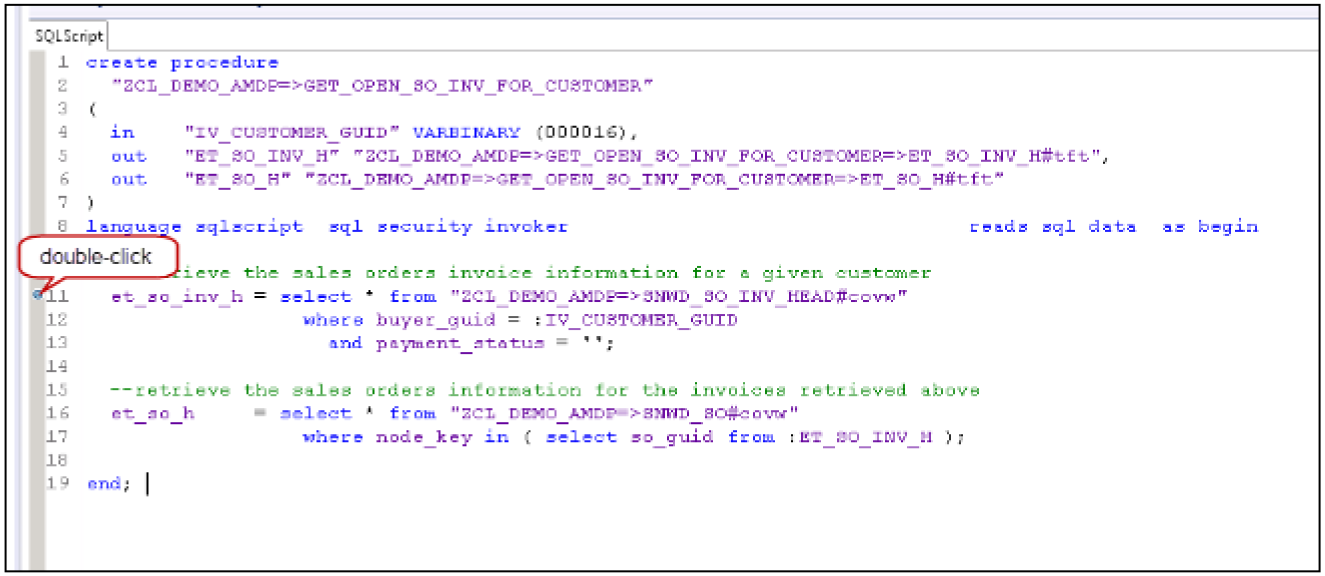


- HDB (DEBUGUSER)
  - Catalog
    - Public Synonyms
    - DEBUGUSER
    - SAPHANAABAP
      - Column Views
      - Functions
      - Indexes
      - Procedures - Filter : *DEMO*
        - Table Types
        - ZCL_DEMO_AMDP=>GET_OPEN_SO_INV_FOR_CUSTOMER
        - ZCL_DEMO_AMDP=>GET_OPEN_SO_INV_FOR_CUSTOMER#stub#2
      - Sequences
      - Synonyms
      - Tables
      - Triggers
      - Views
    - SYS
    - _SYS_REPO
  - Content
  - Provisioning
  - Security

# AMDP Debugging

Set a break point by Debug User in Catalog DB Procedure

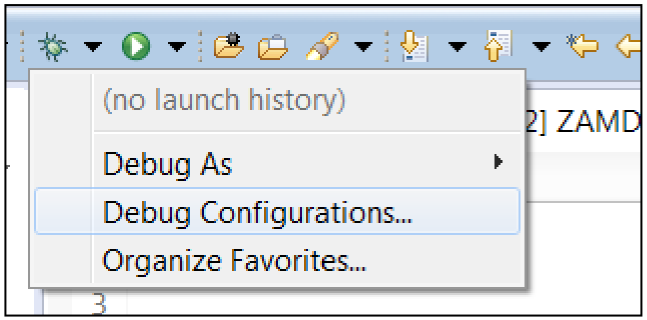
Set a break point by double click on the left section of the line

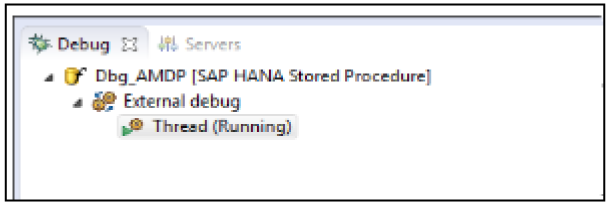


```
create procedure
  "ZCL_DEMO_AMDP=>GET_OPEN_SO_INV_FOR_CUSTOMER"
(
  in     "IV_CUSTOMER_GUID" VARBINARY (000016),
  out    "ET_SO_INV_H" "ZCL_DEMO_AMDP=>GET_OPEN_SO_INV_FOR_CUSTOMER=>ET_SO_INV_H#tft",
  out    "ET_SO_H" "ZCL_DEMO_AMDP=>GET_OPEN_SO_INV_FOR_CUSTOMER=>ET_SO_H#tft"
)
language sqlscript  sql security invoker                              reads sql data  as begin

          ieve the sales orders invoice information for a given customer
  et_so_inv_h = select * from "ZCL_DEMO_AMDP=>SNWD_SO_INV_HEAD#covw"
                where buyer_guid = :IV_CUSTOMER_GUID
                  and payment_status = '';

  --retrieve the sales orders information for the invoices retrieved above
  et_so_h     = select * from "ZCL_DEMO_AMDP=>SNWD_SO#covw"
                where node_key in ( select so_guid from :ET_SO_INV_H );

end;
```

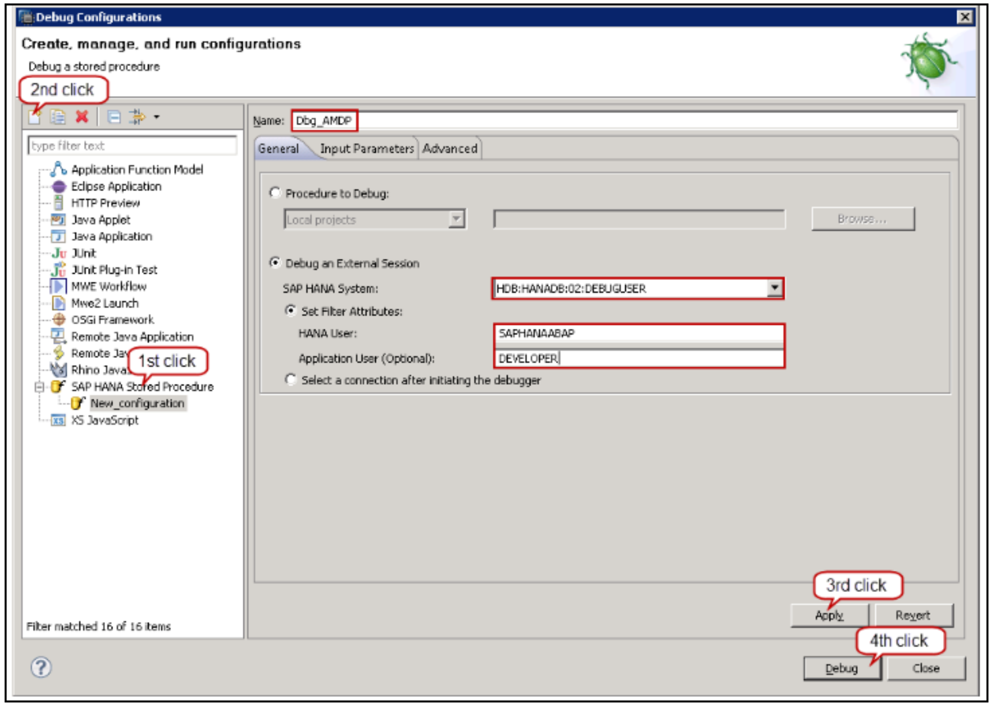
# AMDP Debugging

- Create debug configuration with Filter criteria
- Start SQL script debugger ( Attach session of HANA User )





Dbg_AMDP [SAP HANA Stored Procedure]
External debug
Thread (Running)



Create, manage, and run configurations

Debug a stored procedure

type filter text

- Application Function Model
- Eclipse Application
- HTTP Preview
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- MWE Workflow
- Mwe2 Launch
- OSGi Framework
- Remote Java Application
- Remote Ja
- Rhino Java
- SAP HANA Stored Procedure
  - New_configuration
- XS JavaScript

Filter matched 16 of 16 items

Name: Dbg_AMDP

General | Input Parameters | Advanced

Procedure to Debug:

Local projects | Browse...

Debug an External Session

SAP HANA System: HDB:HANADB:02:DEBUGUSER

Set Filter Attributes:

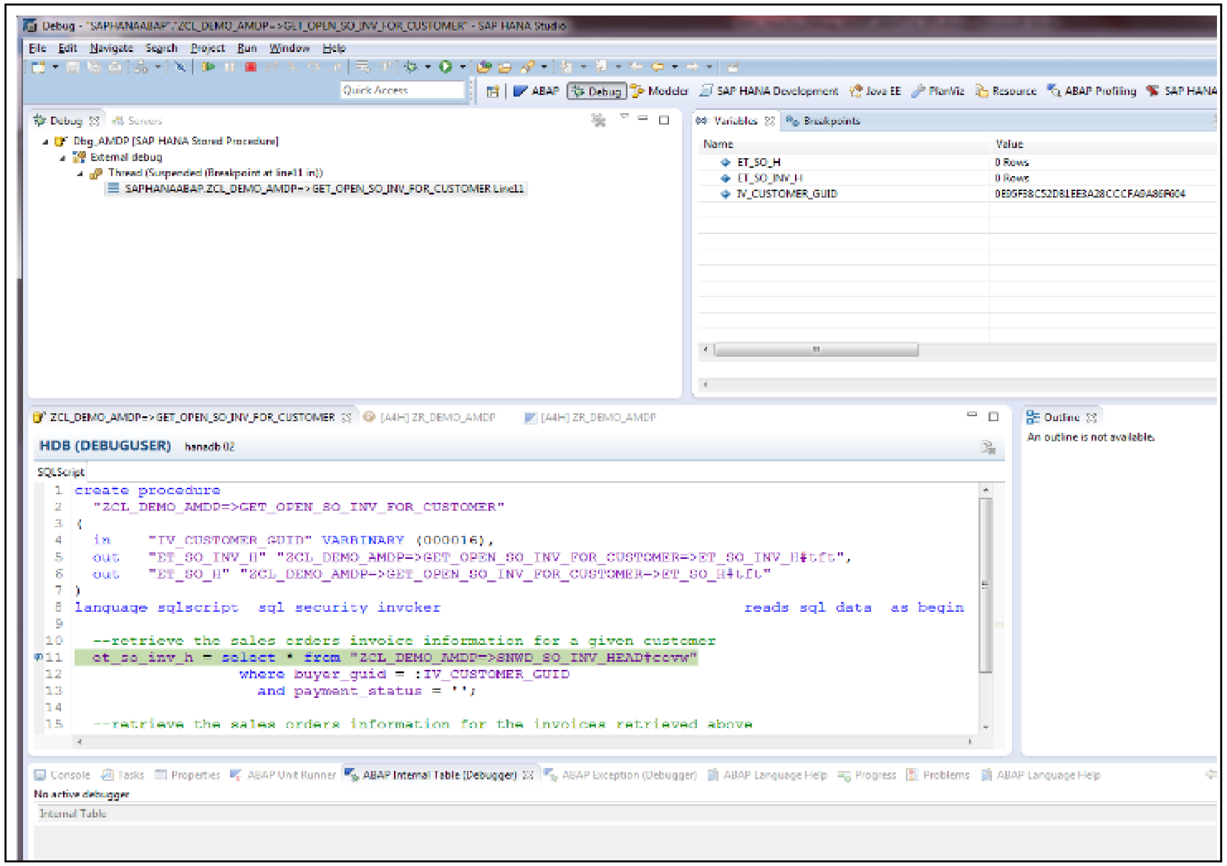
HANA User: SAPHANAABAP

Application User (Optional): DEVELOPER

Select a connection after initiating the debugger

# AMDP Debugging

- Execute the report by Developer user



Debug - "SAPHANAABAP"."ZCL_DEMO_AMDP=>GET_OPEN_SO_INV_FOR_CUSTOMER" - SAP HANA Studio

Dbg_AMDP [SAP HANA Stored Procedure]
External debug
Thread (Suspended (Breakpoint at line11 in))
SAPHANAABAP.ZCL_DEMO_AMDP=>GET_OPEN_SO_INV_FOR_CUSTOMER.Line11

| Name | Value |
|---|---|
| ET_SO_H | 0 Rows |
| ET_SO_INV_H | 0 Rows |
| IV_CUSTOMER_GUID | 0E0GF38C52D81EE8A28CCCFA0A80F604 |

ZCL_DEMO_AMDP=>GET_OPEN_SO_INV_FOR_CUSTOMER

HDB (DEBUGUSER) hanadb 02

SQLScript

```
create procedure
  "ZCL_DEMO_AMDP=>GET_OPEN_SO_INV_FOR_CUSTOMER"
(
  in     "IV_CUSTOMER_GUID" VARBINARY (000016),
  out    "ET_SO_INV_H" "ZCL_DEMO_AMDP=>GET_OPEN_SO_INV_FOR_CUSTOMER=>ET_SO_INV_H#tft",
  out    "ET_SO_H" "ZCL_DEMO_AMDP=>GET_OPEN_SO_INV_FOR_CUSTOMER=>ET_SO_H#tft"
)
language sqlscript  sql security invoker                     reads sql data  as begin

  --retrieve the sales orders invoice information for a given customer
  et_so_inv_h = select * from "ZCL_DEMO_AMDP=>SNWD_SO_INV_HEAD#covw"
                where buyer_guid = :IV_CUSTOMER_GUID
                  and payment_status = '';

  --retrieve the sales orders information for the invoices retrieved above
```

An outline is not available.

No active debugger

## AMDP Debugging

- Required Authorizations - SAP note 1942471

- For the HANA Debug User, the authorization to read the catalogue needs to be granted by the SYSTEM user:

  - **grant catalog** read **to <DEBUG USER>**;

## AMDP Debugging

- The corresponding grant statements to be executed in the SQL console of the SAP HANA studio (as SAP<SID> user) for the ABAP Managed DB procedure <AMDP_NAME> are:

- **grant debug on <HANA USER>."<**AMDP_NAME>" **to** <DEBUG USER> ;
- **grant execute on <HANA USER>."<** AMDP_NAME>" **to** <DEBUG USER>;
- **grant** attach debugger **to** <DEBUG USER>;

## Summary

In this lesson, you have learnt:

- How to use native SQL for SAP HANA
- Different SQL syntaxes used for SAP HANA
- About ABAP Managed Database Procedures (AMDP) and its functionality
- AMDP Debugging

# Review Question

SQLScript is used in SAP HANA when other modeling constructs of HANA such as Attribute views or Analytic views are not sufficient.

- True
- False

The main functionality of AMDP is/are ---------.

Prerequisites of AMDP debugging are

- Set a break point by Debug User in Catalog DB Procedure
- Open SAP developer User schema
- Both

During the execution of AMDP procedures every procedure of the call hierarchy runs either in debug mode or in optimized mode.

- True
- False

Thank you