



SAP HANA

Lesson Name: ABAP Managed
Database Procedures

Lesson Objectives



- After completing this lesson, participants will be able to -
- Understand the benefits of using ABAP-Managed Database Procedures
 - .Create ABAP-managed database procedures
 - .Call ABAP-managed database procedures in ABAP



Procedures define reusable data processing functions.

This means that you can avoid developing overly complex calculation again and again.

You simply define a procedure and call this as and when required.

For example,

You could create a procedure that calculates the tax for each item sold. We would simply define the input parameter as a tax percentage and then define the output parameters as total calculated tax .

The complex logic to calculate the tax will be inside the procedure.

Calling Stored Procedures








As other database systems support stored procedures, calling and creating them in ABAP has long been possible.

Stored procedures are database-specific.

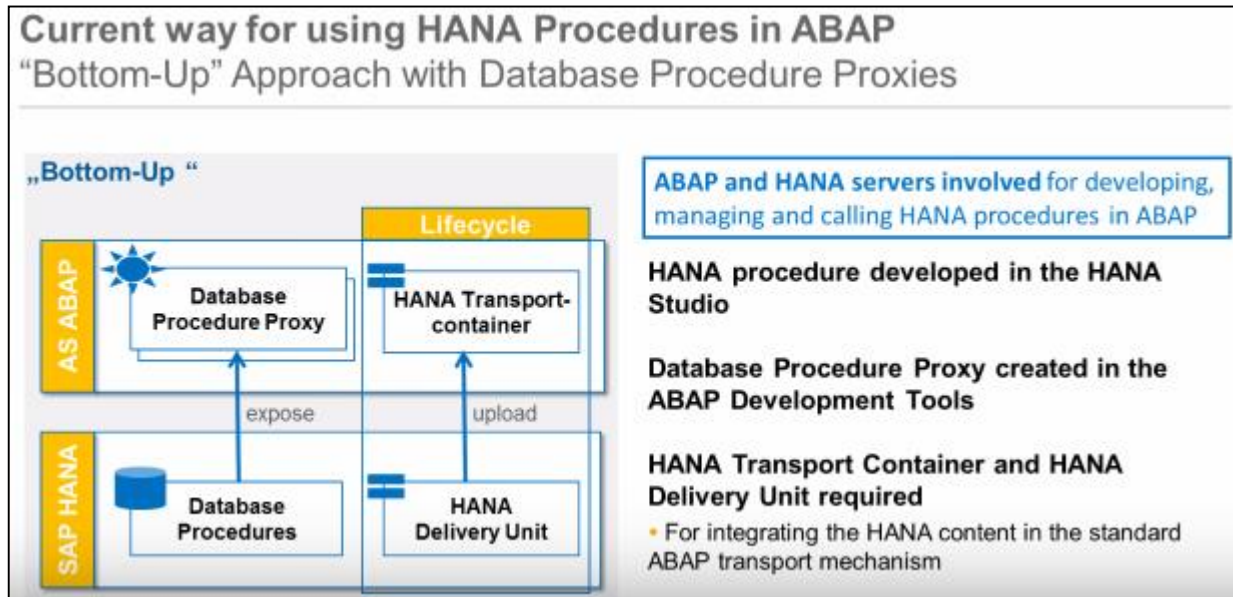
They can be called:

1. Using Native SQL
2. ADBC
3. AMDP

	EXEC SQL	ABAP Database Connectivity API
Call procedure	<pre>EXEC SQL. execute procedure dbproc (in p_in1, inout p_inout1) ENDEXEC.</pre> 	<pre>lr_stmt->set_param(REF #(im_src)). lr_stmt->execute_procedure(`dbproc`).</pre> 
Create procedure	<pre>EXEC SQL. create procedure dbproc(in i_param1 integer, inout ch_param3 nvarchar (20)) language sqlscript sql security invoker as begin select top :i_param1 * from sys.dummy; ch_param3 := 'Success'; end; ENDEXEC.</pre> 	<pre>l_ddl_command = create procedure dbproc(& in i_param1 integer, & inout ch_param3 nvarchar (20)) & language sqlscript & sql security invoker as begin & select top :i_param1 * & from sys.dummy; & ch_param3 := 'Success'; end; . im_sql_handle->execute_ddl(l_ddl_command).</pre> 

 Figure 112: Using Stored Procedures in ABAP < 7.40

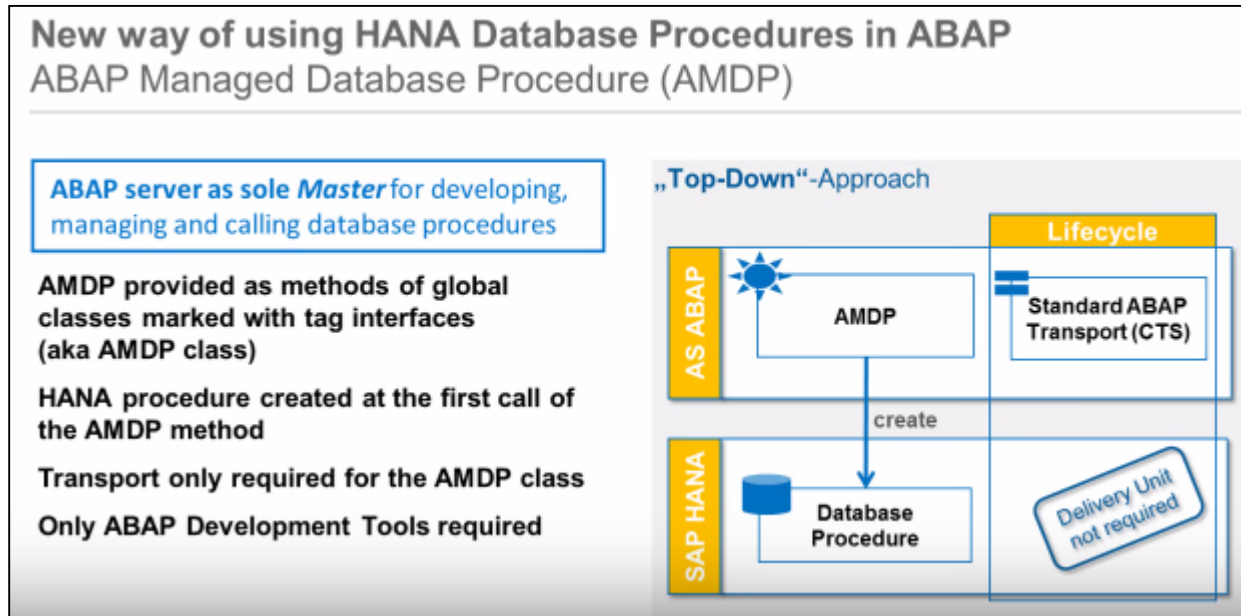
Classical approach of using Procedures



In the Bottom-up approach, the procedure has first to be developed in HANA server and then exposed the **Database Procedure Proxy** in **ABAP server**.

The optimized way for using HANA procedures in ABAP is provided by Database Procedure Proxies which was introduced with Application ABAP 7.4 with service package 2

New approach of using Procedures :AMDP



The Top-Down approach enables developers to create a **procedure** in ABAP Development Environment.

The AMDP is implemented as a method of a global class which is marked with specific interfaces called as AMDP class.

In corresponding to AMDP class, the **HANA based SQL** class is created at the first call of the method.

What is AMDP?



AMDP stands for ABAP-Managed Database Procedures

AMDP is a new feature in ABAP allowing developers to write database procedures directly in ABAP.

Database Procedure is a function stored and executed in the database.

The implementation language varies from one database system to another.

In SAP HANA it is SQL Script.

Using AMDP allows developers to create and execute those database procedures in the ABAP environment using ABAP methods and ABAP data types.

AMDP is specifically for HANA Database.



- The basic idea of AMDP is to manage HANA procedures and their lifecycle inside the ABAP server.
- To allow native consumption of HANA features from within the ABAP layer, the HANA database procedure language SQLScript has been integrated into the ABAP stack.
- AMDP is implemented in ABAPclass methods (so-called AMDP methods) that serve as a container for SQLScript code.
- This approach enables the shipment of AMDP in the same way as any other ABAP development object (lifecycle management)

Code-to-Data Paradigm

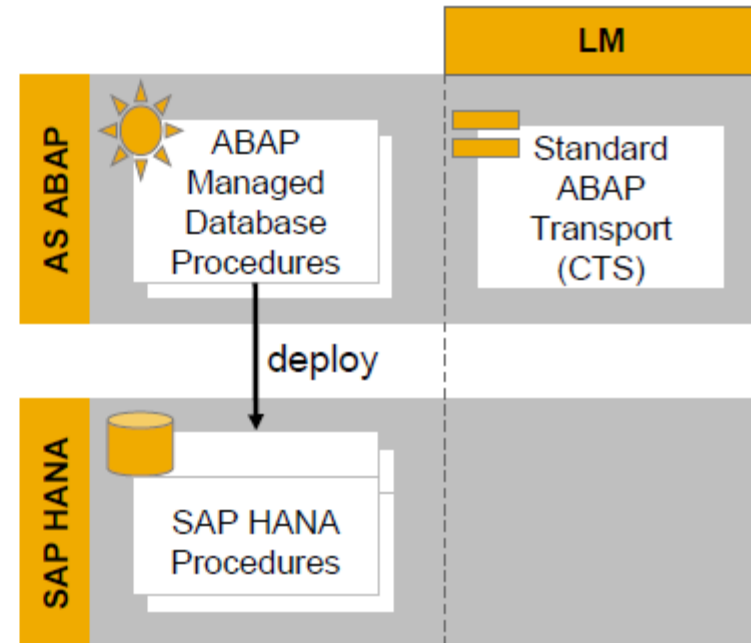
- Supported through embedding native database procedure coding

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





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.
    
```



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 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)  
    ).
```

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(1x).
        "do some meaningful error handling
ENDTRY.
```



• 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 Debugging

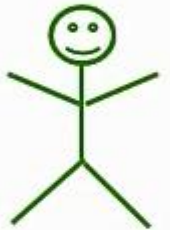


For AMDP Debugging, we need 3 users.

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



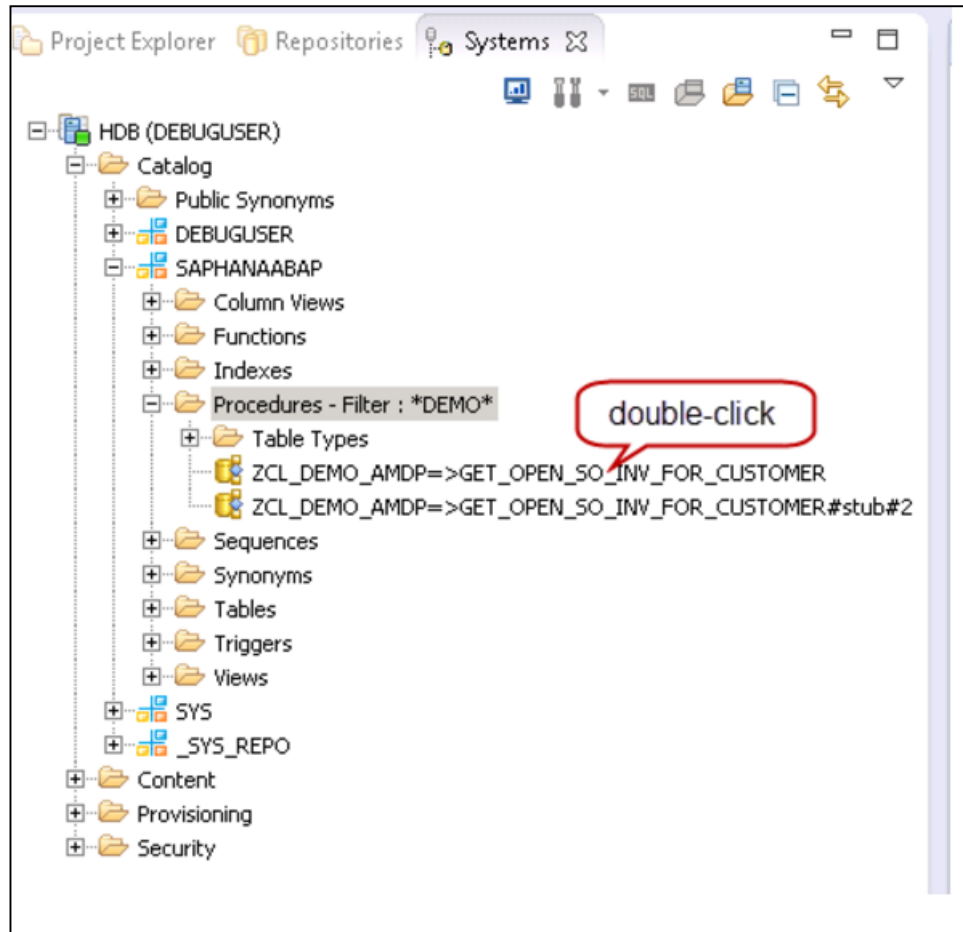


- Set a break point by Debug User in Catalog DB Procedure
- Create debug configuration with Filter criteria
- Start SQL script debugger (Attach session of HANA User)
- Developer now execute the ABAP report and AMDP called



- 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



AMDP Debugging



Set a break point by Debug User in Catalog DB Procedure

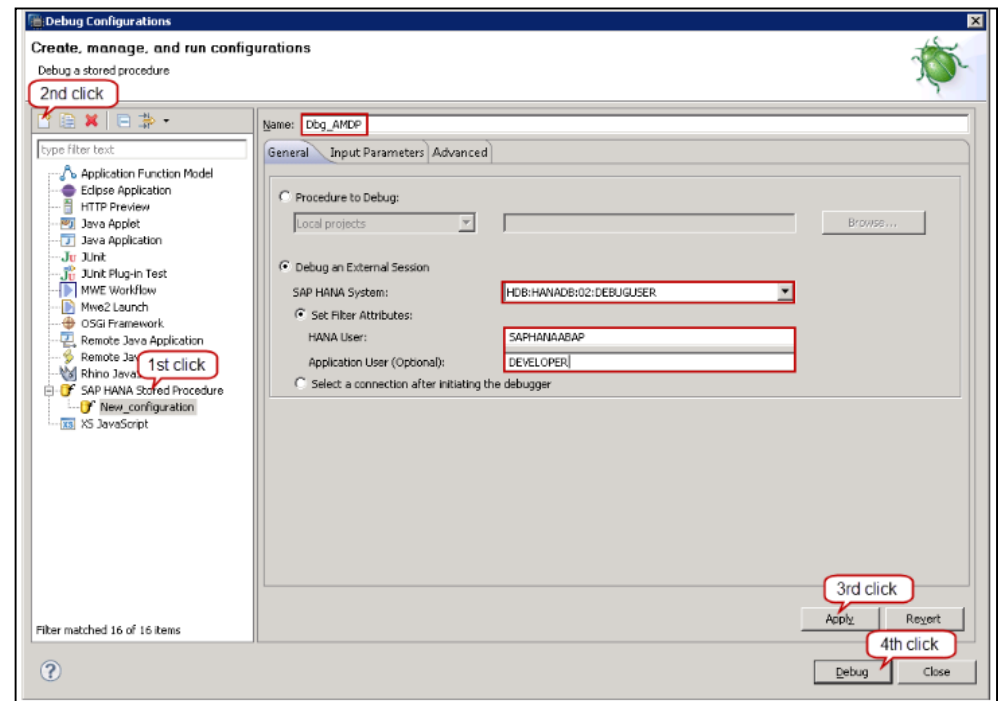
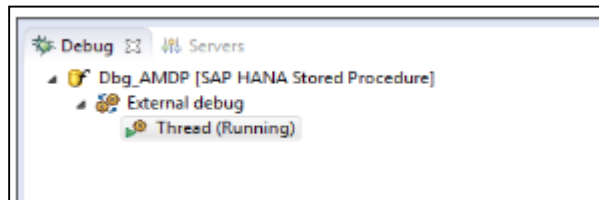
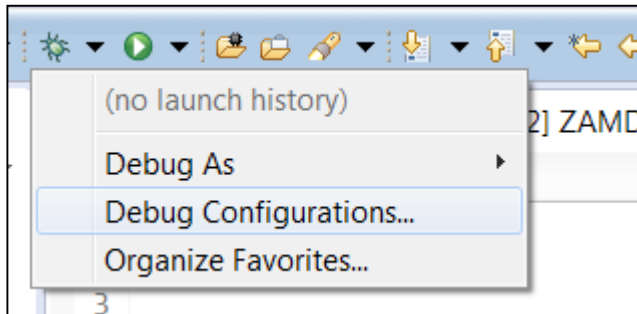
Set a break point by double click on the left section of the line

```
SQLScript
1 create procedure
2   "ZCL_DEMO_AMDP=>GET_OPEN_SO_INV_FOR_CUSTOMER"
3 {
4   in   "IV_CUSTOMER_GUID" VARBINARY (000016),
5   out  "ET_SO_INV_H" "ZCL_DEMO_AMDP=>GET_OPEN_SO_INV_FOR_CUSTOMER=>ET_SO_INV_H#tft",
6   out  "ET_SO_H" "ZCL_DEMO_AMDP=>GET_OPEN_SO_INV_FOR_CUSTOMER=>ET_SO_H#tft"
7 }
8 language sqlscript sql security invoker reads sql data as begin
9   --retrieve the sales orders invoice information for a given customer
11  et_so_inv_h = select * from "ZCL_DEMO_AMDP=>SNWD_SO_INV_HEAD#covw"
12                  where buyer_guid = :IV_CUSTOMER_GUID
13                  and payment_status = '';
14
15  --retrieve the sales orders information for the invoices retrieved above
16  et_so_h      = select * from "ZCL_DEMO_AMDP=>SNWD_SO#covw"
17                  where node_key in ( select so_guid from :ET_SO_INV_H );
18
19 end; |
```

AMDP Debugging



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



AMDP Debugging



- Execute the report by Developer user

The screenshot displays the SAP HANA Studio interface during a debug session. The top toolbar includes standard development actions like Run, Stop, and Step Through. The left pane shows the project structure with the procedure 'ZCL_DEMO_AMDP->GET_OPEN_SO_INV_FOR_CUSTOMER' selected. The right pane, titled 'Variables', shows the current state of variables: 'ET_SO_H' (0 Rows), 'ET_SO_INV_H' (0 Rows), and 'IV_CUSTOMER_GUID' (DE25F8C52D81EE3A28CCCFA0A8CF004). The main editor shows the SQLScript code for the procedure, which includes a comment about retrieving sales orders invoice information. The bottom status bar indicates 'No active debugger'.

```
1 create procedure
2   "ZCL_DEMO_AMDP->GET_OPEN_SO_INV_FOR_CUSTOMER"
3 {
4   in   "IV_CUSTOMER_GUID" VARBINARY (000016),
5   out  "ET_SO_INV_H" "ZCL_DEMO_AMDP->GET_OPEN_SO_INV_FOR_CUSTOMER->ET_SO_INV_H",
6   out  "ET_SO_H" "ZCL_DEMO_AMDP->GET_OPEN_SO_INV_FOR_CUSTOMER->ET_SO_H"
7 }
8 language sqlscript sql security invoker reads sql data as begin
9
10  --retrieve the sales orders invoice information for a given customer
11  et_so_inv_h = select * from "ZCL_DEMO_AMDP->SNWD_SO_INV_HEAD+ccvw"
12    where buyer_guid = :IV_CUSTOMER_GUID
13    and payment_status = '';
14
15  --retrieve the sales orders information for the invoices retrieved above
```



- 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>;**



- 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:

- About ABAP Managed Database Procedures (AMDP) and its functionality
- AMDP Debugging

Review Question



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