Modern ABAP Questions

This document only includes answers to key questions that were not discussed during the live course.

Week 3: ABAP-Managed Database Procedures (AMDP) and SQLScript

The answers to questions asked during Week 3 of Modern ABAP (*ABAP-Managed Database Procedures (AMDP) and SQLScript*), held on September 24, 2024, are presented here. In some instances, multiple questions about the same topic have been combined (as denoted by the tag "cluster topic").

1. Cluster topic: AMDP methods versus ABAP methods

AMDP methods shouldn't be evaluated as regular ABAP methods. An AMDP method is simply a wrapper for a SQLScript procedure. For each AMDP method, a corresponding SQLScript procedure is created on the SAP HANA database—and there is no concept such as "ABAP class" there. When our AMDP method calls another AMDP method, we actually make one SQLScript procedure call another SQLScript procedure.

That's why we can't access global class variables from within AMDP methods, and that's why we have to pass all parameters by value (instead of by reference). SQLScript code within the AMDP method is nearly blind to the ERP environment, it can only access input output parameters passed from ABAP.

2. Cluster topic: Dynamic SQL

Regarding `APPLY_FILTER`: I have never encountered a limit regarding the size of the filter string. It makes sense though that it would share an overall limit with dynamic SQL clauses, as `APPLY_FILTER` contains a dynamic condition as well.

Here is a resource for SQL injection prevention: *https://help.sap.com/docs/hana-cloud-database/sap-hana-cloud-sap-hana-sqlscript-reference/sql-injection-prevention-functions*.

3. Using AMDP, what is the best way to expose a service to pull data via SQLScript?

Current technology to host a service would be the ABAP RESTful application programming model (frequently called "RAP"), which is CDS-based. During Week 2 (CDS) of this course series, we learned that we have two options to bind an ABAP class behind a CDS: virtual elements or custom entities. In either case, you have an ABAP class to fill the CDS, where you can write SQLScript any way you like.

Developing an AMDP-supported CDS table function and consuming it in your RAP CDS may be another approach.

In the ABAP programming model for SAP Fiori, your function behind SEGW can call an ABAP class containing an AMDP to build the result set.

In the classic ABAP world without any bells and whistles, it is possible to host RESTful services over raw BSP, which will receive HTTP `GET` or `POST` requests along parameters. You can make your BSP page call an ABAP class containing an AMDP method.

4. What about text elements for translations?

SQLScript is mostly blind to the SAP ERP environment, so you have to plan your texts accordingly. In some cases, you can use the variable `SESSION_CONTEXT('LOCALE_SAP')` to access the current language, and read tables like MAKT, T100, etc. accordingly. In more complex cases, such as SE91 or SO10 content or program text elements, you can simply leave the text stuff to ABAP code.

Text elements are rarely performance bottlenecks anyway.

The sample class `CL_DEMO_HANA_SESSION_VARIABLES` contains examples of other `SESSION_CONTEXT` variables. In case you can't find this class, you can read: *https://help.sap.com/doc/abapdocu_752_index_htm/7.52/en-US/abenhana_session_variables_abexa.htm*.

5. Cluster topic: AMDP exceptions

ABAP constants can't be accessed by the AMDP method directly, simply because it runs on a different platform (SAP HANA database). It can only access what you pass as a variable, or what is stored in a table. Instead of using constants, you can store those values in a table—that way, SQLScript can access them and use to them to build dynamic error messages.

For error message texts: How about storing them in SE91? You can surely access T100 from SQLScript and read the appropriate message text using the logon language available in `SESSION_CONTEXT('LOCALE_SAP')`.

Regarding AMDP exception handling, here are two excellent blog posts, which go beyond our introduction to the subject:

- *https://sapcodes.com/2020/07/03/hana-sql-signal-resignal/*
- *https://sapcodes.com/2020/07/01/hana-sql-exit-handler/*

If AMDP exception handling seems too complex, you can simply export an error code from the AMDP method and raise the corresponding exception using ABAP.

6. Is there a concept of "clean" a temp table when I do not need it anymore in the rest of the code?

Not directly, but here is a workaround:

```
:tmp_table = select * from :tmp_table where 1 = 2;
```

7. What is a better approach, to use that row_number window function or user a group by clause?

They serve different purposes. `ROW_NUMBER` serves the purpose of ranking rows and possibly selecting top N rows from those ranks. `GROUP BY` serves the purpose of grouping rows for aggregation. They are not alternatives to each other.

8. **I have come across the term "CREATE PROCEDURE" a few times in connection with SQLScript, what is the meaning of it?**

   If you bypass the ERP system completely and log into SAP HANA using an appropriate tool (Eclipse), you would have to use the command `CREATE PROCEDURE` to create a SQLScript procedure, just like other similar database products.

   But as an ERP programmer using ABAP, you don't have to. When you create an AMDP method, SAP takes care of creating a corresponding stored procedure in the background, possibly using a dynamic `CREATE PROCEDURE` statement.

9. **While inserting/updating records this way, how can we ensure data consistency like we can do with update task FMs?**

   You can take advantage of `COMMIT` and `ROLLBACK` in SQLScript, as in any decent database environment. Even `SAVEPOINT` is supported, in case you have an exceptionally complex situation. More info here: *https://help.sap.com/docs/SAP_HANA_PLATFORM/de2486ee947e43e684d39702027f8a94/e4e1b570e4f04b93ae1538246894b496.html*.