AMDP: Avoiding FOR ALL ENTRIES and pushing calculation to Database Layer

June 19, 2018 | 876 Views |

Ankit Rastogi
more by this author

SAP S/4HANA

ABAP Development | amdp | CALCULATION IN DB LAYER | for all entries | group by | inner join with internal table | range table | RANGE TABLE CROSS REFERENCE

1. Objective

The objective of this document is to explain step-by-step process to create AMDP method using multiple select queries to avoid FOR ALL ENTRIES and push calculation to database layer.

2. Requirement

Requirement is to fetch records from database table MATDOC based on certain plant and storage location combination. On the fetched records, perform calculation e.g. summation on quantity based on various combinations e.g. Material/ Plant/Storage Location, Material/Plant, Material. The developer would like to leverage AMDP to address this requirement.

Relevant fields of MATDOC Table:

Fields	Type	Key
WERKS	WERKS_D	
MATNR	MATNR	
LGORT	LGORT_D	
ERFMG	ERFMG	

Input Tables:

List of Materials

Fields	Type	Key
MATNR	MATNR	

List of Plant and Storage Location combination

Fields	Type	Key
WERKS	WERKS_D	

LGORT	LGORT_D	

Output Tables:

Fields	Type	Key
MATNR	MATNR	
WERKS	WERKS_D	
ERFMG (SUM)	ERFMG	

3. Understanding limitation in FOR ALL ENTRIES select statement

In a select query, with FOR ALL ENTRIES, one can't use Group BY clause. The addition GROUP BY has no effect if FOR ALL ENTRIES is used.

With new directive of S/4 HANA coding, all the calculation should be pushed to database layer. Hence one can't leverage the code pushdown if FOR ALL ENTRIES is used in select query.

To avoid FOR ALL ENTRIES in select query, one can go ahead and use multiple ranges for each field of driver table of select query. But with multiple ranges, we get cross referencing entries.

1. Range Table cross referencing entries

Plant	**Storage Location**	**Number of Entries in MATDOC with Plant/Storage Location combination**	**Number of Entries in MATDOC when both Plant/Storage location are passed as individual ranges**
0001	0001	412	
1010	0002	0	

SUM	412	460

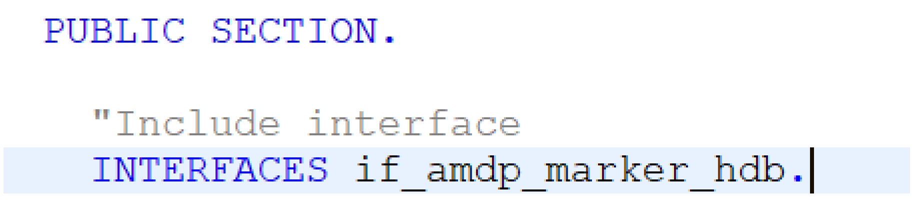
As you can see number of entries are considerably increased because of cross referencing of plant and storage location i.e. Plant 0001 & Storage location 0002 combination AND Plant 1010 & Storage location 0002 combination is fetching extra (458 – 412 = 48) Entries.

4. Configuration

The following steps explain step by step configuration:

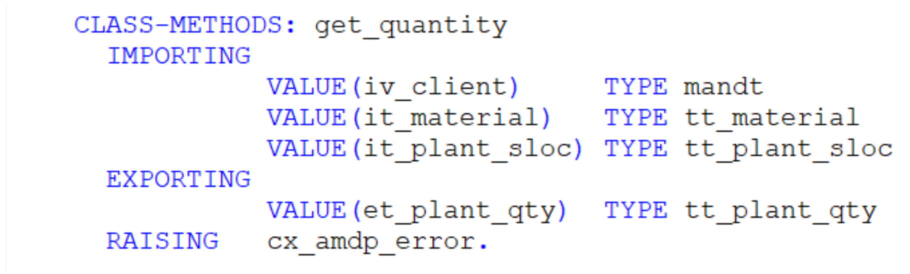
1. Create an AMDP Method inside a class

Include the IF_AMDP_MARKER_HDB interface in the class. See below screenshot.



```
PUBLIC SECTION.

  "Include interface
  INTERFACES if_amdp_marker_hdb.
```

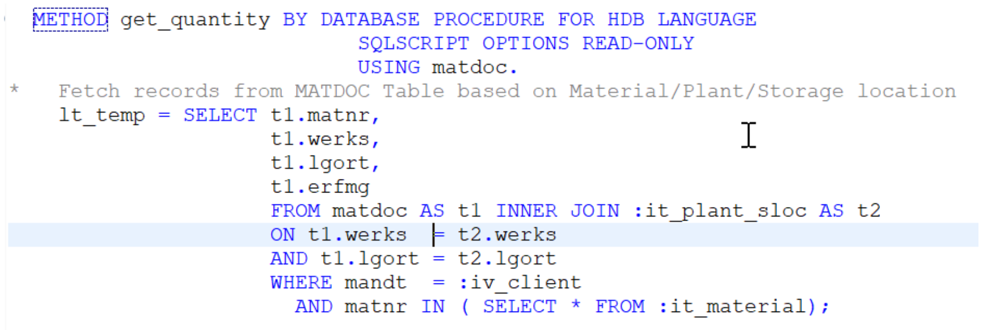
Define the method as below screenshot. Input parameters include list of materials and list of plant and storage locations.



```
CLASS-METHODS: get_quantity
  IMPORTING
            VALUE(iv_client)     TYPE mandt
            VALUE(it_material)   TYPE tt_material
            VALUE(it_plant_sloc) TYPE tt_plant_sloc
  EXPORTING
            VALUE(et_plant_qty)  TYPE tt_plant_qty
  RAISING   cx_amdp_error.
```

2. Write first select statement

Prepare first select statement based on list of materials and list of plant & storage locations. See below screenshot. Pay attention to AMDP method implementation syntax.

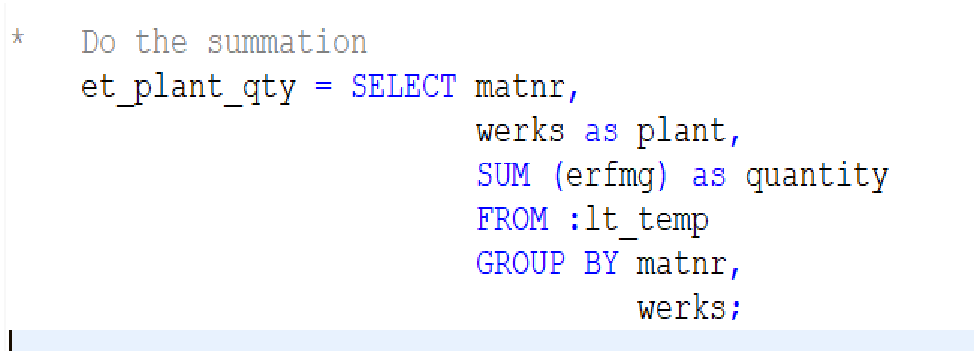


```
METHOD get_quantity BY DATABASE PROCEDURE FOR HDB LANGUAGE
                          SQLSCRIPT OPTIONS READ-ONLY
                          USING matdoc.
*   Fetch records from MATDOC Table based on Material/Plant/Storage location
    lt_temp = SELECT t1.matnr,
                     t1.werks,
                     t1.lgort,
                     t1.erfmg
                     FROM matdoc AS t1 INNER JOIN :it_plant_sloc AS t2
                     ON t1.werks  = t2.werks
                     AND t1.lgort = t2.lgort
                     WHERE mandt  = :iv_client
                       AND matnr IN ( SELECT * FROM :it_material);
```

Here we have used inner join on database table with input parameter table.

3. Write subsequent select statement

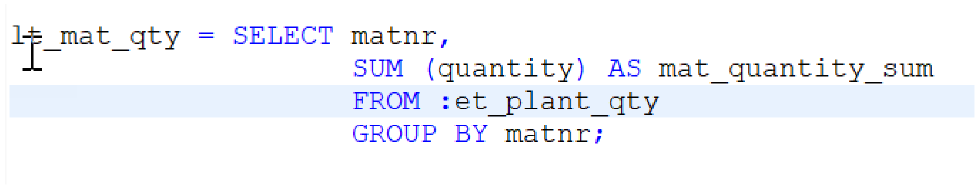
One good feature of AMDP is that one can write select statements on local variables e.g. local internal tables. Write second select statement on records fetched in 1st select statement and use GROUP BY clause.



```
*   Do the summation
    et_plant_qty = SELECT matnr,
                          werks as plant,
                          SUM (erfmg) as quantity
                          FROM :lt_temp
                          GROUP BY matnr,
                                   werks;
```

4. Use GROUP BY clause in resulting dataset

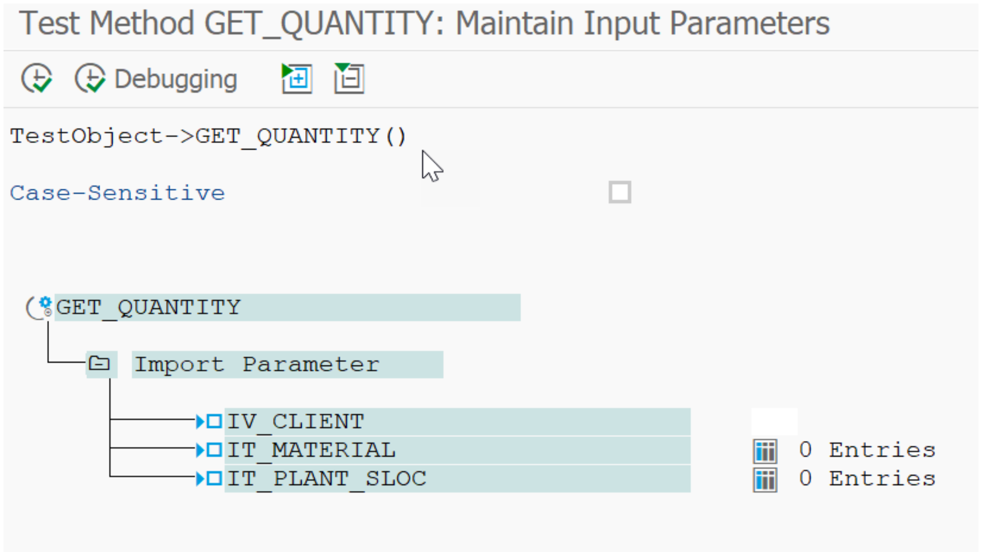
Now when we have resulting dataset, we can write further select statements on local internal table obtained in 1st select statement with various conditions of GROUP BY class. This will enable us to perform quantity summation (calculation) and prepare output in desired format. One can write multiple select statements based on requirements. See below screenshot.



```
lt_mat_qty = SELECT matnr,
                    SUM (quantity) AS mat_quantity_sum
                    FROM :et_plant_qty
                    GROUP BY matnr;
```

5. Test

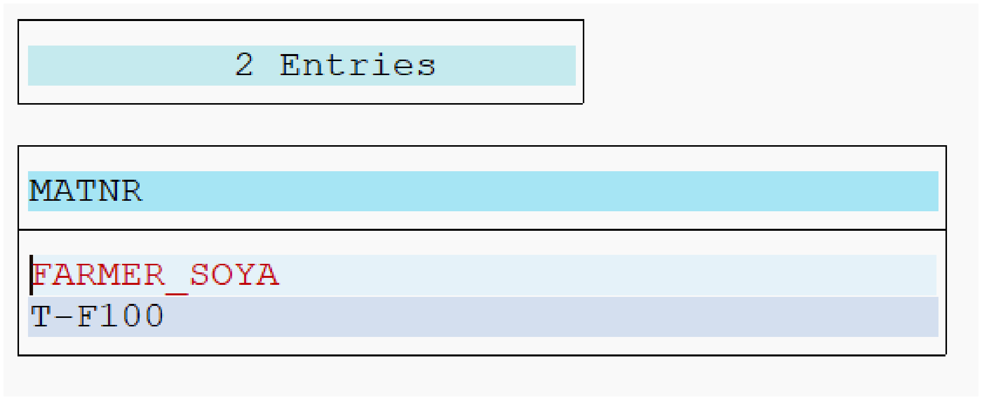
Now run the AMDP method by executing class from SE24 transaction. It should open the window to test the method.



Test Method GET_QUANTITY: Maintain Input Parameters

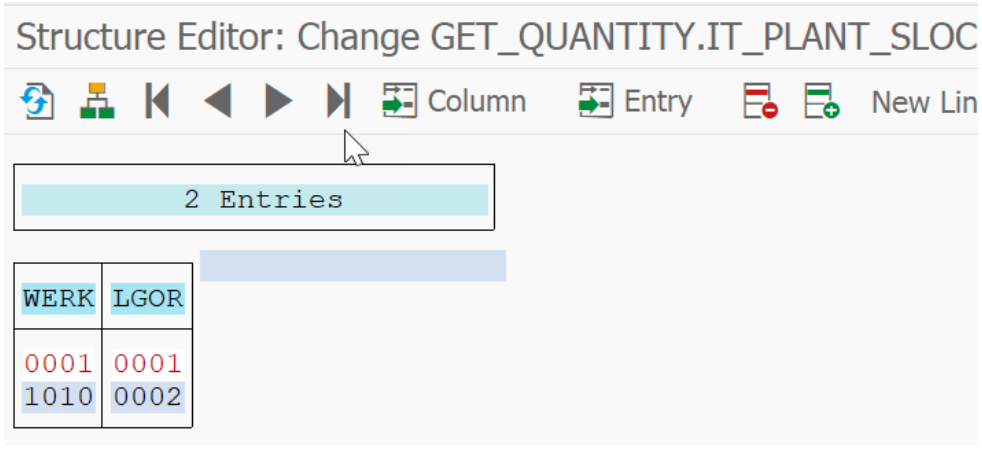
```
TestObject->GET_QUANTITY()

Case-Sensitive

GET_QUANTITY
  Import Parameter
    IV_CLIENT
    IT_MATERIAL          0 Entries
    IT_PLANT_SLOC        0 Entries
```

Populate the Material List, Plant List and Storage Location List as below



2 Entries

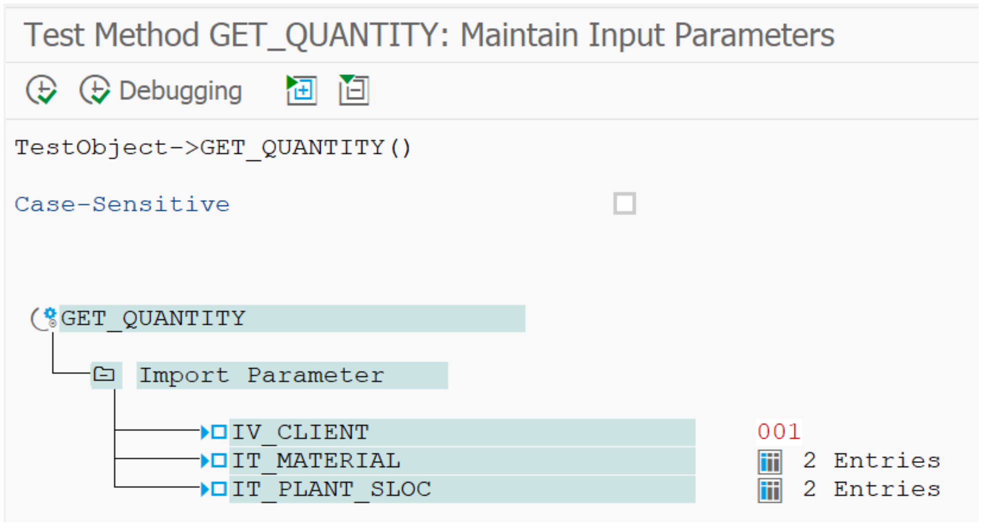
MATNR
FARMER_SOYA
T-F100



Structure Editor: Change GET_QUANTITY.IT_PLANT_SLOC

2 Entries

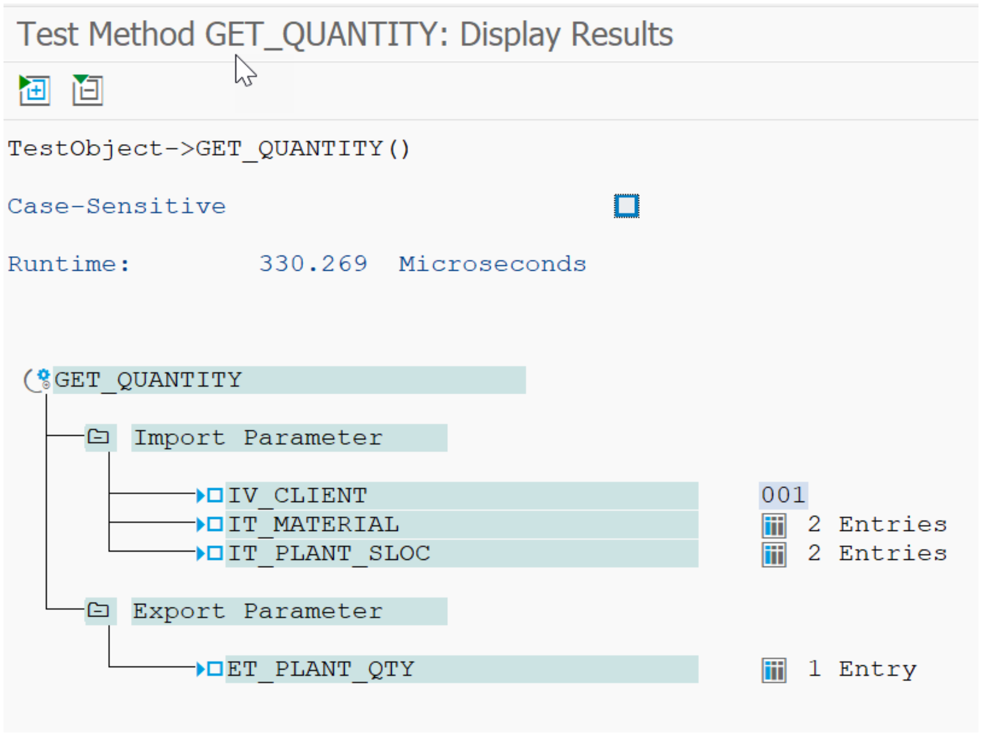
WERK	LGOR
0001	0001
1010	0002


Test Method GET_QUANTITY: Maintain Input Parameters

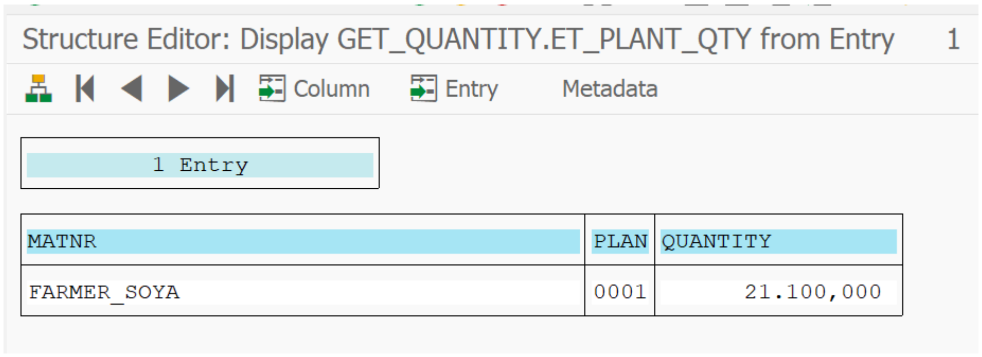
Debugging

```
TestObject->GET_QUANTITY()

Case-Sensitive

GET_QUANTITY
  Import Parameter
    IV_CLIENT                          001
    IT_MATERIAL                        2 Entries
    IT_PLANT_SLOC                      2 Entries
```

Press execute button and see the result in export parameter table ET_PLANT_QTY


Test Method GET_QUANTITY: Display Results

```
TestObject->GET_QUANTITY()

Case-Sensitive

Runtime:          330.269   Microseconds

GET_QUANTITY
  Import Parameter
    IV_CLIENT                          001
    IT_MATERIAL                        2 Entries
    IT_PLANT_SLOC                      2 Entries
  Export Parameter
    ET_PLANT_QTY                       1 Entry
```


Structure Editor: Display GET_QUANTITY.ET_PLANT_QTY from Entry 1

Column Entry Metadata

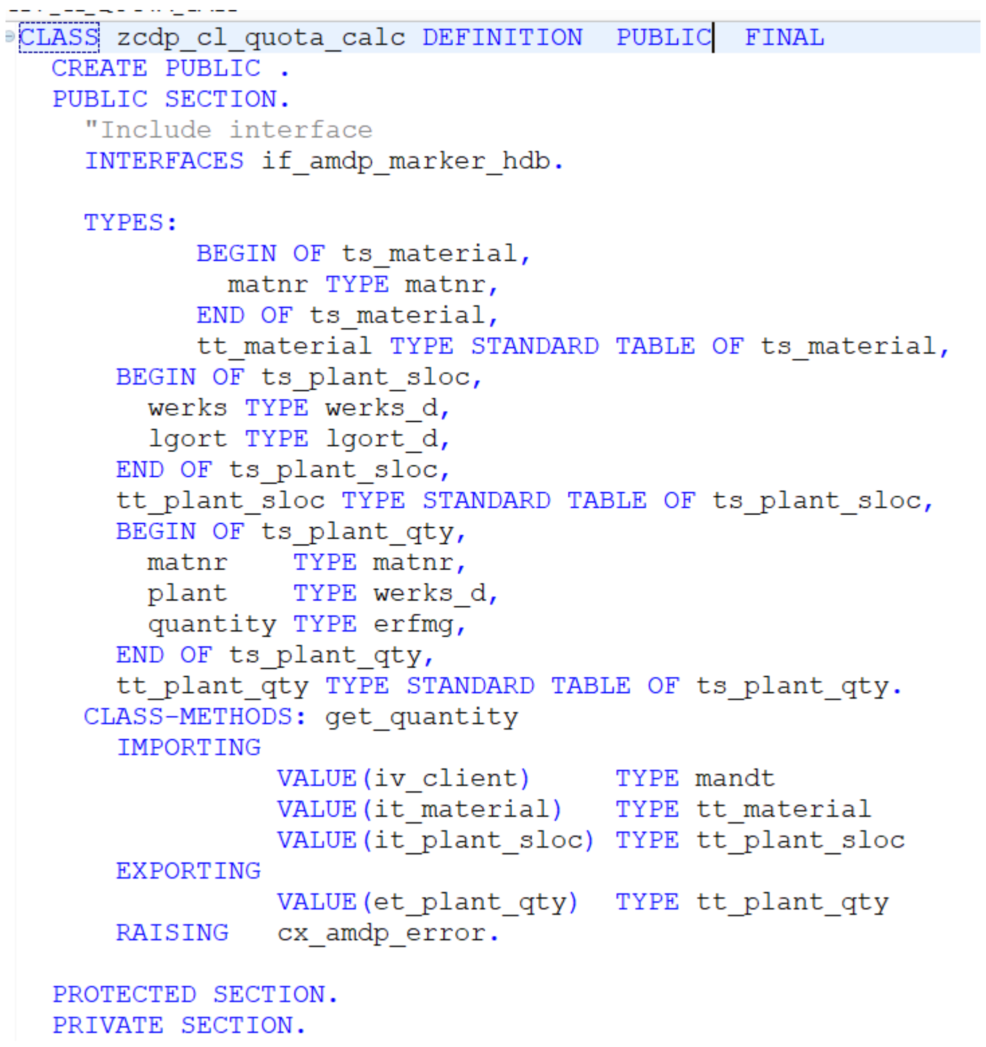
1 Entry

MATNR	PLAN	QUANTITY
FARMER_SOYA	0001	21.100,000

6. Coding

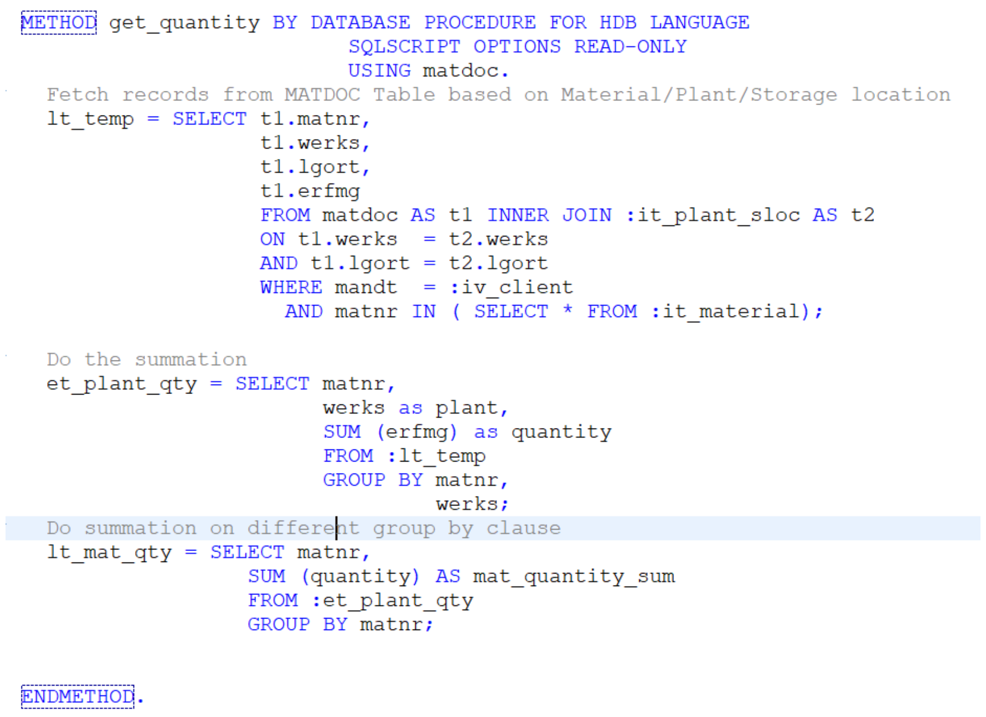
Coding part follows standard SQL Script references. Here select statement is broken into multiple steps depending upon select options.

See below screenshot for Class/Method definition



```
CLASS zcdp_cl_quota_calc DEFINITION  PUBLIC  FINAL
  CREATE PUBLIC .
  PUBLIC SECTION.
    "Include interface
    INTERFACES if_amdp_marker_hdb.

    TYPES:
           BEGIN OF ts_material,
             matnr TYPE matnr,
           END OF ts_material,
           tt_material TYPE STANDARD TABLE OF ts_material,
      BEGIN OF ts_plant_sloc,
        werks TYPE werks_d,
        lgort TYPE lgort_d,
      END OF ts_plant_sloc,
      tt_plant_sloc TYPE STANDARD TABLE OF ts_plant_sloc,
      BEGIN OF ts_plant_qty,
        matnr    TYPE matnr,
        plant    TYPE werks_d,
        quantity TYPE erfmg,
      END OF ts_plant_qty,
      tt_plant_qty TYPE STANDARD TABLE OF ts_plant_qty.
    CLASS-METHODS: get_quantity
      IMPORTING
                VALUE(iv_client)     TYPE mandt
                VALUE(it_material)   TYPE tt_material
                VALUE(it_plant_sloc) TYPE tt_plant_sloc
      EXPORTING
                VALUE(et_plant_qty)  TYPE tt_plant_qty
      RAISING   cx_amdp_error.

  PROTECTED SECTION.
  PRIVATE SECTION.
```

See below screenshot for Method implementation.



```
METHOD get_quantity BY DATABASE PROCEDURE FOR HDB LANGUAGE
                          SQLSCRIPT OPTIONS READ-ONLY
                          USING matdoc.
   Fetch records from MATDOC Table based on Material/Plant/Storage location
   lt_temp = SELECT t1.matnr,
                    t1.werks,
                    t1.lgort,
                    t1.erfmg
                    FROM matdoc AS t1 INNER JOIN :it_plant_sloc AS t2
                    ON t1.werks  = t2.werks
                    AND t1.lgort = t2.lgort
                    WHERE mandt  = :iv_client
                      AND matnr IN ( SELECT * FROM :it_material);

   Do the summation
   et_plant_qty = SELECT matnr,
                         werks as plant,
                         SUM (erfmg) as quantity
                         FROM :lt_temp
                         GROUP BY matnr,
                                  werks;
   Do summation on different group by clause
   lt_mat_qty = SELECT matnr,
                   SUM (quantity) AS mat_quantity_sum
                   FROM :et_plant_qty
                   GROUP BY matnr;

ENDMETHOD.
```

7. Limitation

All standard limitations of AMDP such as:

1. An AMDP class can only be edited in ADT (Eclipse).
2. Client will not be handled automatically like in open SQL.
3. In case of CDS Views, write appropriate annotations in CDS View definition for client handling so that they can be used inside AMDP. Accordingly, AMDP definition will change.
4. Exposed associations in CDS Views can’t be accessed inside AMDP.
5. As of now, AMDP only works when underlying database is HANA.

Alert Moderator

8 Comments

You must be Logged on to comment or reply to a post.

Gaurav Sharan

June 19, 2018 at 10:24 am

Cool…Straight forward example.. ! Well Done !!

Mehmet Dagnilak

June 20, 2018 at 9:04 am

I loved the ideas of joining with an internal table and selecting from an local internal table with grouping. Thank you very much for pointing these!

I wonder how Hana handles selecting from an internal table. Does it read all the data into the internal table and then summarize it, or does it do all the calculations in one step?

Ankit Rastogi Post author

June 25, 2018 at 4:59 am

Hi Mehmet

I guess it would be similar to any other DB tables i.e. fetch all the data then do the summation.

Thanks

Ankit

Zhe Zhao

June 21, 2018 at 5:38 am

With ABAP 7.52, an internal table can be specified as a data source data source of a query. So we can join an internal table with datebase table directly by using Open SQL. However, it is not possible to use more than one internal table in an SQL command.

Mehmet Dagnilak

June 25, 2018 at 6:40 am

I loved this even better 🙂 I wish companies could upgrade more often..

Ankit Rastogi   Post author

June 25, 2018 at 4:57 am

Thanks Zhe for letting us know on that.

Sijin Chandran

September 19, 2018 at 7:26 am

Hi Ankit,

First very thanks for this informative blog.

Am pretty new to AMDP coding, I have one question,

While implementing AMDP Methods like below:

```
METHOD get_kna1_catalog BY DATABASE PROCEDURE
                FOR HDB
                LANGUAGE SQLSCRIPT
                OPTIONS  READ-ONLY
                USING kna1.
```

can’t we have more than one Table reference in the using part (highlighted below).

```
USING kna1
```

In my case in the same method I want to fetch from various related tables like knvv, knvp etc and at this part am not able to mention more than one table.

Helpful views much appreciated.

Thanks,

Sijin

Sijin Chandran

September 20, 2018 at 1:16 pm

Found the answer,

Yes we can :

https://help.sap.com/doc/abapdocu_751_index_htm/7.51/en-US/abenamdp_functions_abexa.htm