// Explore More Tutorials

Analyze AMDP Performance

Julie Plummer 03/11/2019

Beginner 45 min. SAP Cloud Platform, ABAP environment, Tutorial, SAP Cloud Platform, Beginner, ABAP Development

Analyze the runtime performance of AMDPs and the executed SQL statements using the AMDP Profiler in ABAP Development Tools (ADT).

You will learn

- How to create an ABAP class containing an ABAP Managed Database Procedure (`AMDP`)
- How to run the ABAP Profiler on this class

Provide Feedback

Throughout this tutorial, objects name include the suffix **XXX**. Always replace this with your group number or initials.

You should be familiar with ABAP Managed Database Procedures (AMDP). Briefly, AMDP allows you to optimize your ABAP code (for ABAP on SAP HANA) by calling HANA database procedures from a global ABAP class.For more details, see:

- SAP Help Portal: ABAP Managed Database Procedures (AMDP)
- ABAP Keyword Documentation: AMDP - ABAP Managed Database Procedures

1 Step 1: Install the Flight Reference Scenario using abapGit

1. Create the package `/DMO/FLIGHT` as a sub-package under the package `/DMO/SAP` (keep the default values). **IMPORTANT:** Make sure that the software component is also `/DMO/SAP`.

2. Open the `abapGit` view by choosing **Window > Show View > Other... > `abapGit` Repositories**. Make sure you have the correct ABAP Cloud Project marked (See the little headline in the `abapGit` view for the current project.)

3. Clone a repository by choosing **+** and enter the repository URL: https://github.com/SAP/abap-platform-refscen-flight.git.

4. On the next page choose the master branch and provide the package `/DMO/FLIGHT`.

5. Create or assign a transport request and choose **Finish**. This starts the cloning of the repository, which might take a few minutes

6. Once the cloning has finished, refresh your project tree.

7. Finally, generate some sample data by running the ABAP class `/DMO/CL_FLIGHT_DATA_GENERATOR` by choosing **Run as Console Application (F9)**.

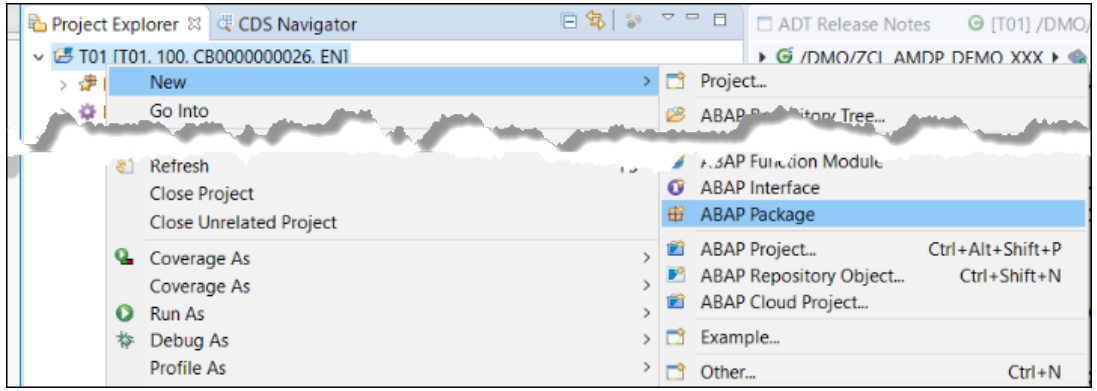
Done

Log on to answer question

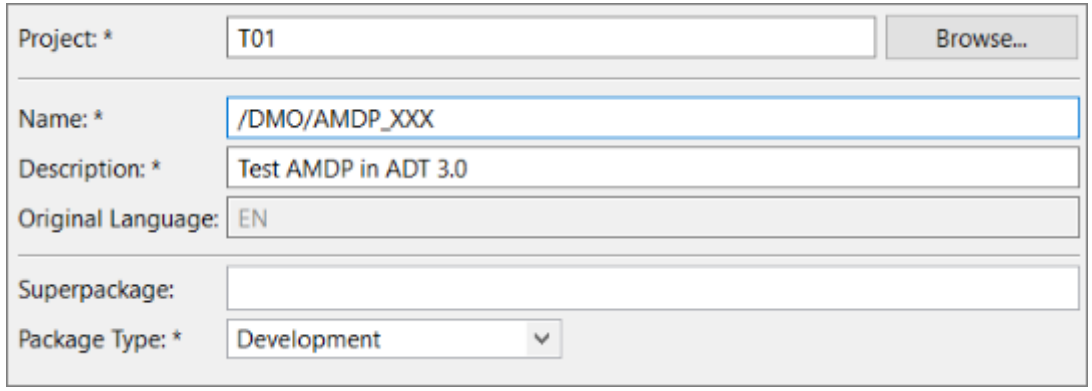
2 Step 2: Create an ABAP package in /DMO/SAP

One of the restrictions of the ABAP Environment on SAP Cloud Platform is that you can only use other objects if they are released objects (whitelisted) or in the same software component. Since you will be using objects in the `/DMO/SAP` software component, you need to create your package in `/DMO/SAP`.

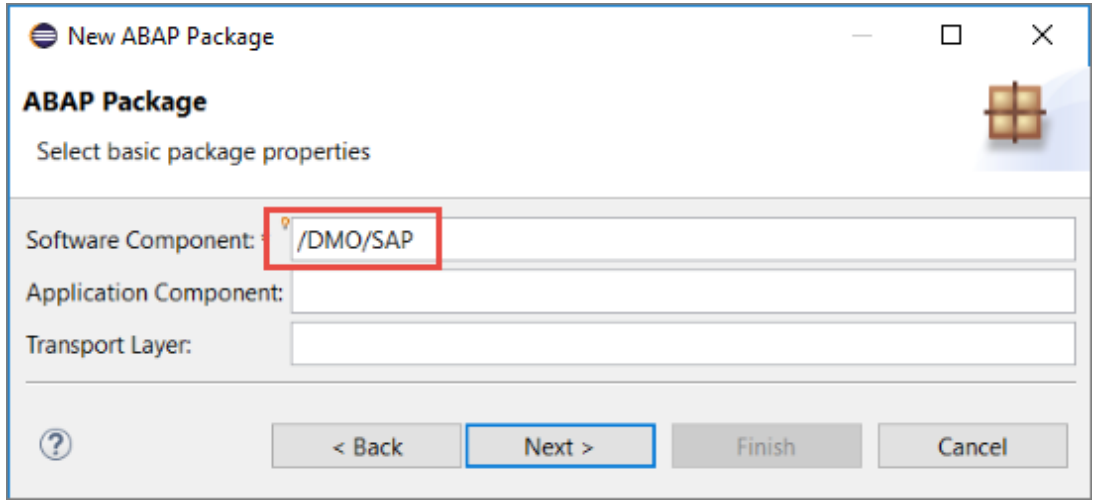
1. Select your project. From the context menu, choose **New > ABAP Package**.



2. Enter a name and description for your package, then choose **Next**.**IMPORTANT**: Make sure that you prefix the name with **`/DMO/`**, for example: Name = `/DMO/AMDP_XXX` (replacing **`XXX`** with your group number or initials)



3. Enter the software component `/DMO/SAP`. You can leave the other fields blank.



4. Create or assign a transport request and choose **Finish**.

Done

Log on to answer question

3 Step 3: Create an ABAP class

1. Again, from the context menu of your package, choose **New > ABAP Class**.
2. Enter a name and description, then choose **Next**. Make sure that you prefix the class name with `/DMO/`, for example:
    - Name: `/DMO/ZCL_AMDP_DEMO_XXX`
    - Description: AMDP Demo w Flight ref

3. Assign the transport request and choose **Finish**.

Done

Log on to answer question

4 Step 4: Add two interfaces

Add two interfaces by adding this code to the public section.

- **`if_amdp_marker_hdb`** defines the class as an AMDP class, allowing you to implement AMDP methods - that is, ABAP methods that call a SAP HANA database procedure from within a global ABAP class.
- **`if_oo_adt_classrun`** allows you to output the results to the ABAP Console.

ABAP Copy

```abap
INTERFACES: if_amdp_marker_hdb,
if_oo_adt_classrun.

```

Done

Log on to answer question

5 Step 5: Create structures and table types

Add these structures and types to the public section, just after the interface definitions.Note the data elements that you imported earlier.

ABAP Copy

```
TYPES:
  BEGIN OF ty_result_line,
    airline           TYPE /dmo/carrier_name,
    flight_connection TYPE /dmo/connection_id,
    old_price         TYPE /dmo/flight_price,
    old_currency      TYPE /dmo/currency_code,
    new_price         TYPE /dmo/flight_price,
    new_currency      TYPE /dmo/currency_code,
  END OF ty_result_line,

  BEGIN OF ty_flights_line,
    airline           TYPE /dmo/carrier_name,
    flight_connection TYPE /dmo/connection_id,
    price             TYPE /dmo/flight_price,
    currency          TYPE /dmo/currency_code,
  END OF ty_flights_line,

  ty_result_table  TYPE STANDARD TABLE OF ty_result_line WITH EMPTY
  ty_flights_table TYPE STANDARD TABLE OF ty_flights_line WITH EMPTY
  ty_flights       TYPE STANDARD TABLE OF /dmo/flight.

```

Done

Log on to answer question

6 Step 6: Add method definitions

Add these two method definitions to your code. Ignore the errors for now.

ABAP Copy

```
METHODS:
  get_flights
    EXPORTING
              VALUE(result) TYPE ty_result_table
    RAISING   cx_amdp_execution_error,

  convert_currency
    IMPORTING
              VALUE(flights) TYPE ty_flights_table
    EXPORTING
              VALUE(result)  TYPE ty_result_table
    RAISING   cx_amdp_execution_error.


```

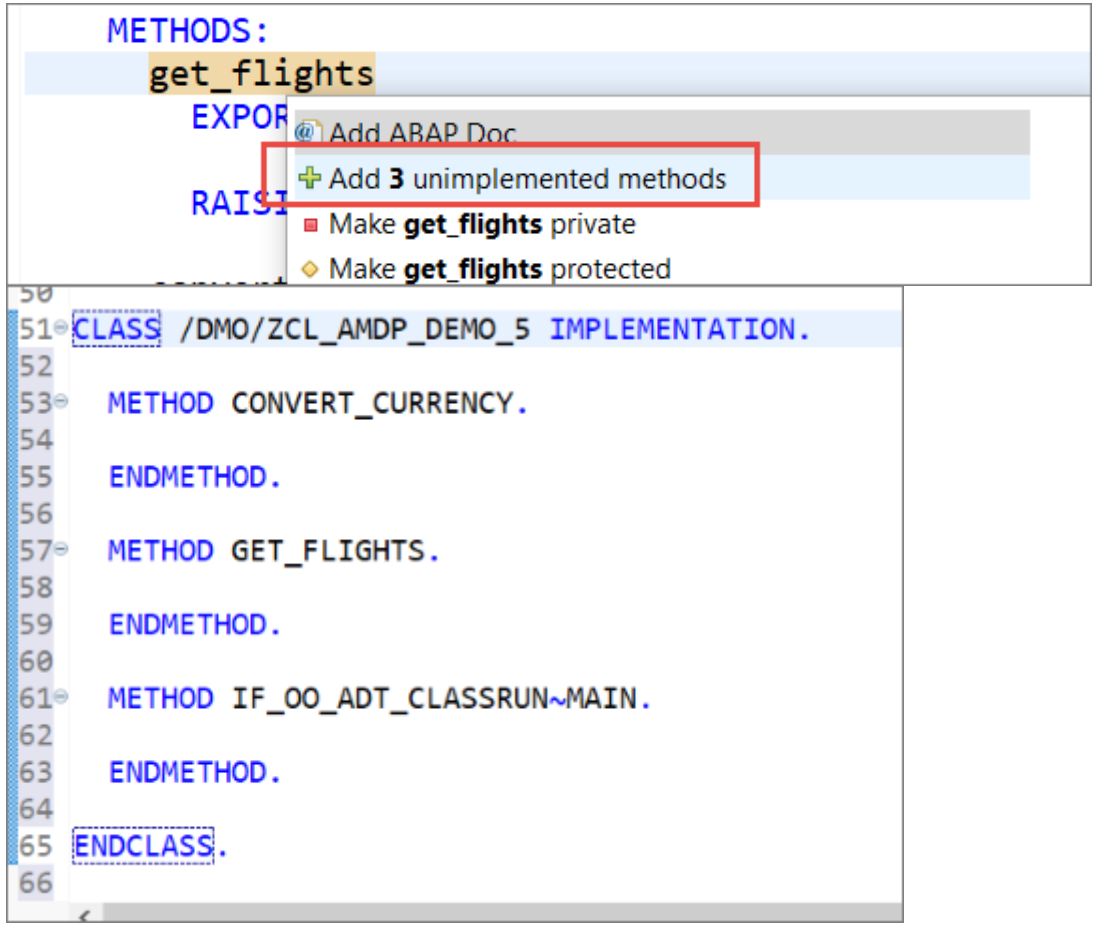
Both of these are AMDP methods.

Done

Log on to answer question

7 Step 7: Implement get_flights

1. In the class definition, select any one of the methods and choose **Add 3 unimplemented methods**. All three (empty) implementations will appear in the class implementation.



2. Add the following to the method `get_flights`. Both this and `convert_currency` are `SQLScript`.

   For more information on SAP HANA `SQLScript`, see SAP HANA SQLScript Reference

ABAP Copy

```
METHOD GET_FLIGHTS by database procedure
  for hdb
```

```
  language sqlscript
  options read-only
  using
    /dmo/flight
    /dmo/carrier
    /dmo/zcl_amdp_demo_xxx=>convert_currency.

```

You must specify all ABAP tables, views, and AMDP procedures in the USING clause.For more details on these clauses, see ABAP Keyword Documentation: Method - By Database Procedure, Function

3. Add the SELECT statement. Ignore the warning for now.

ABAP Copy

```
flights = select distinct

  from "/DMO/FLIGHT"  as f
  inner join "/DMO/CARRIER" as c
    on f.carrier_id = c.carrier_id;

```

4. Add the fields. You can do this using Auto-complete (`Ctrl+1`), to make sure you are using the correct field names.

ABAP Copy

```
flights = select distinct
  c.name as airline,
  f.connection_id as flight_connection,
```

```
  f.price    as price,
  f.currency_code as currency
  from "/DMO/FLIGHT"  as f
  inner join "/DMO/CARRIER" as c
    on f.carrier_id = c.carrier_id;

```

5. Finally, call the other AMDP method, `convert_currency`. Your method should look like this:

ABAP Copy

```
METHOD get_flights BY DATABASE PROCEDURE
  FOR HDB
  LANGUAGE SQLSCRIPT
  OPTIONS READ-ONLY
  USING
    /dmo/flight
    /dmo/carrier
    /dmo/zcl_amdp_demo_xxx=>convert_currency.

  flights = select distinct
    c.name as airline,
    f.connection_id as flight_connection,
    f.price    as price,
    f.currency_code as currency
    from "/DMO/FLIGHT"  as f
    inner join "/DMO/CARRIER" as c
      on f.carrier_id = c.carrier_id;

  call "/DMO/ZCL_AMDP_DEMO_XXX=>CONVERT_CURRENCY"( :flights, re

```

```
ENDMETHOD.
```

Done

Log on to answer question

8 Step 8: Implement the method convert_currency

Similarly, implement the `convert_currency` method.

ABAP Copy

```
METHOD convert_currency BY DATABASE PROCEDURE
  FOR HDB
  LANGUAGE SQLSCRIPT
  OPTIONS READ-ONLY.

  declare today date;
  declare new_currency nvarchar(3);

  select current_date into today from dummy;
  new_currency := 'EUR';

  result = select distinct
```

```
    airline,
    flight_connection,
    price    as old_price,
    currency as old_currency,
    convert_currency(
      "AMOUNT"          => price,
      "SOURCE_UNIT"     => currency,
      "TARGET_UNIT"     => :new_currency,
      "REFERENCE_DATE"  => :today,
      "CLIENT"          => '100',
      "ERROR_HANDLING"  => 'set to null',
      "SCHEMA"          => current_schema
    ) as new_price,
    :new_currency as new_currency
    from :flights;

ENDMETHOD.

```

Done

Log on to answer question

9 Step 9: Implement the method `main` of the interface if_oo_adt_classrun

Finally, implement the `main` method of the interface `if_oo_adt_classrun`. This will allow you to output your results to the ABAP Console.

1. Call the method `get_flights` from the current instance of the class:

ABAP Copy

```
me->get_flights(
  IMPORTING
    result = DATA(lt_result) ).
```

2. Output the result to the console:

```
out->write( lt_result ).
```

3. Wrap this in an exception:

ABAP Copy

```
TRY.

CATCH cx_amdp_execution_error INTO DATA(lx_amdp).
  out->write( lx_amdp->get_longtext( ) ).

```

Your method should now look like this:

ABAP Copy

```
METHOD if_oo_adt_classrun~main.

  TRY.
      me->get_flights(
        IMPORTING
          result = DATA(lt_result) ).

    CATCH cx_amdp_execution_error INTO DATA(lx_amdp).
      out->write( lx_amdp->get_longtext( ) ).
  ENDTRY.

  out->write( lt_result ).

ENDMETHOD.

```

Done

Log on to answer question

10 Step 10: Check your code

Your code should look like this.

ABAP Copy

```
CLASS /dmo/zcl_amdp_demo_xxx DEFINITION
  PUBLIC
```

```
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.
    INTERFACES: if_amdp_marker_hdb,
      if_oo_adt_classrun.

    TYPES:
      BEGIN OF ty_result_line,
        airline           TYPE /dmo/carrier_name,
        flight_connection TYPE /dmo/connection_id,
        old_price         TYPE /dmo/flight_price,
        old_currency      TYPE /dmo/currency_code,
        new_price         TYPE /dmo/flight_price,
        new_currency      TYPE /dmo/currency_code,
      END OF ty_result_line,

      BEGIN OF ty_flights_line,
        airline           TYPE /dmo/carrier_name,
        flight_connection TYPE /dmo/connection_id,
        price             TYPE /dmo/flight_price,
        currency          TYPE /dmo/currency_code,
      END OF ty_flights_line,

      ty_result_table  TYPE STANDARD TABLE OF ty_result_line WITH EM
      ty_flights_table TYPE STANDARD TABLE OF ty_flights_line WITH E
      ty_flights       TYPE STANDARD TABLE OF /dmo/flight.

    METHODS:
      get_flights
        EXPORTING
                  VALUE(result) TYPE ty_result_table
        RAISING   cx_amdp_execution_error,

```

```
37        convert_currency
38          IMPORTING
39                    VALUE(flights) TYPE ty_flights_table
40          EXPORTING
41                    VALUE(result)  TYPE ty_result_table
42          RAISING   cx_amdp_execution_error.
43
44    PROTECTED SECTION.
45    PRIVATE SECTION.
46  ENDCLASS.
47
48
49  CLASS /dmo/zcl_amdp_demo_xxx IMPLEMENTATION.
50
51    METHOD get_flights BY DATABASE PROCEDURE
52      FOR HDB
53      LANGUAGE SQLSCRIPT
54      OPTIONS READ-ONLY
55      USING
56        /dmo/flight
57        /dmo/carrier
58        /dmo/zcl_amdp_demo_xxx=>convert_currency.
59
60      flights = select distinct
61        c.name as airline,
62        f.connection_id as flight_connection,
63        f.price    as price,
64        f.currency_code as currency
65        from "/DMO/FLIGHT"  as f
66        inner join "/DMO/CARRIER" as c
67          on f.carrier_id = c.carrier_id;
68
69      call "/DMO/ZCL_AMDP_DEMO_XXX=>CONVERT_CURRENCY"( :flights, resul
70
```

```
ENDMETHOD.


METHOD convert_currency BY DATABASE PROCEDURE
  FOR HDB
  LANGUAGE SQLSCRIPT
  OPTIONS READ-ONLY.

  declare today date;
  declare new_currency nvarchar(3);

  select current_date into today from dummy;
  new_currency := 'EUR';

  result = select distinct
    airline,
    flight_connection,
    price    as old_price,
    currency as old_currency,
    convert_currency(
      "AMOUNT"          => price,
      "SOURCE_UNIT"     => currency,
      "TARGET_UNIT"     => :new_currency,
      "REFERENCE_DATE"  => :today,
      "CLIENT"          => '100',
      "ERROR_HANDLING"  => 'set to null',
      "SCHEMA"          => current_schema
    ) as new_price,
    :new_currency as new_currency
    from :flights;

ENDMETHOD.


```

```
  METHOD if_oo_adt_classrun~main.

    TRY.
        me->get_flights(
          IMPORTING
            result = DATA(lt_result) ).

      CATCH cx_amdp_execution_error INTO DATA(lx_amdp).
        out->write( lx_amdp->get_longtext( ) ).
    ENDTRY.

    out->write( lt_result ).

  ENDMETHOD.
ENDCLASS.

```

Done

Log on to answer question

11 Step 11: Save, activate, and test your code

1. Save and activate your code by choosing **`Ctrl+S, Ctrl+3`**.
2. Optional: Test your class by running it the ABAP Console (**`F9`**).

Your output should look like this:

Global Class | Class-relevant Local Types | Local Types | Test Classes | Macros

Problems | Properties | Templates | Bookmarks | Task Repositories | Task List | Feed Reader | Transport Organizer | Console

ABAP Console

Table

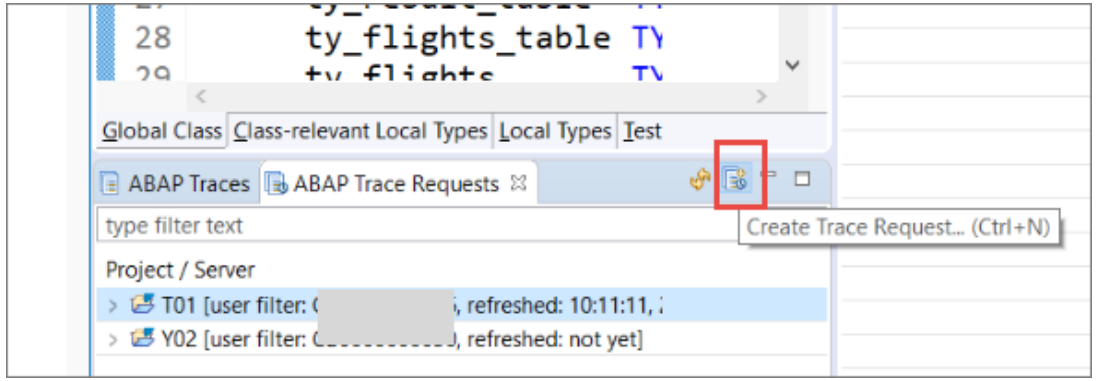
AIRLINE	FLIGHT_CONNECTION	OLD_PRICE	OLD_CURRENCY	NEW_PRICE	NEW_CURRENCY
Singapore Airlines Limited	0001	10818.0	SGD	6596.34	EUR
Singapore Airlines Limited	0001	5950.0	SGD	3628.04	EUR
Singapore Airlines Limited	0002	11765.0	SGD	7173.78	EUR
Singapore Airlines Limited	0002	10953.0	SGD	6678.65	EUR
Singapore Airlines Limited	0011	2359.0	SGD	1438.41	EUR
Singapore Airlines Limited	0011	4880.0	SGD	2975.6	EUR
Singapore Airlines Limited	0012	4665.0	SGD	2844.51	EUR
Singapore Airlines Limited	0012	2574.0	SGD	1569.51	EUR
United Airlines, Inc.	0058	6629.0	USD	7052.12	EUR
United Airlines, Inc.	0058	4996.0	USD	5314.89	EUR
United Airlines, Inc.	0059	4131.0	USD	4394.68	EUR
United Airlines, Inc.	0059	6053.0	USD	6439.36	EUR

Done

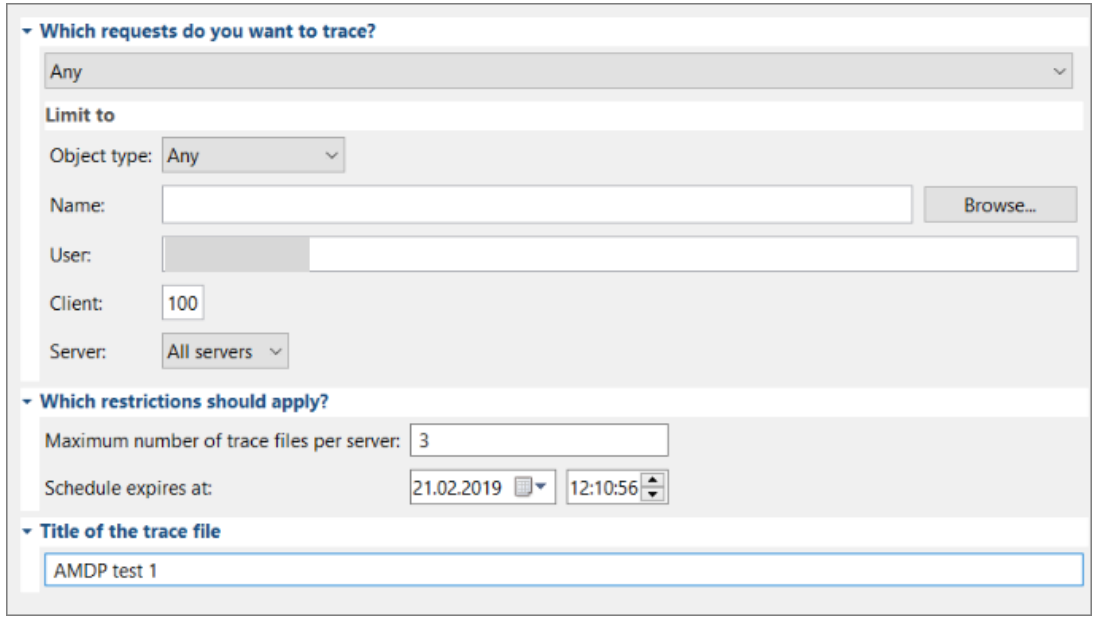
Log on to answer question

12 Step 12: Create an ABAP trace request

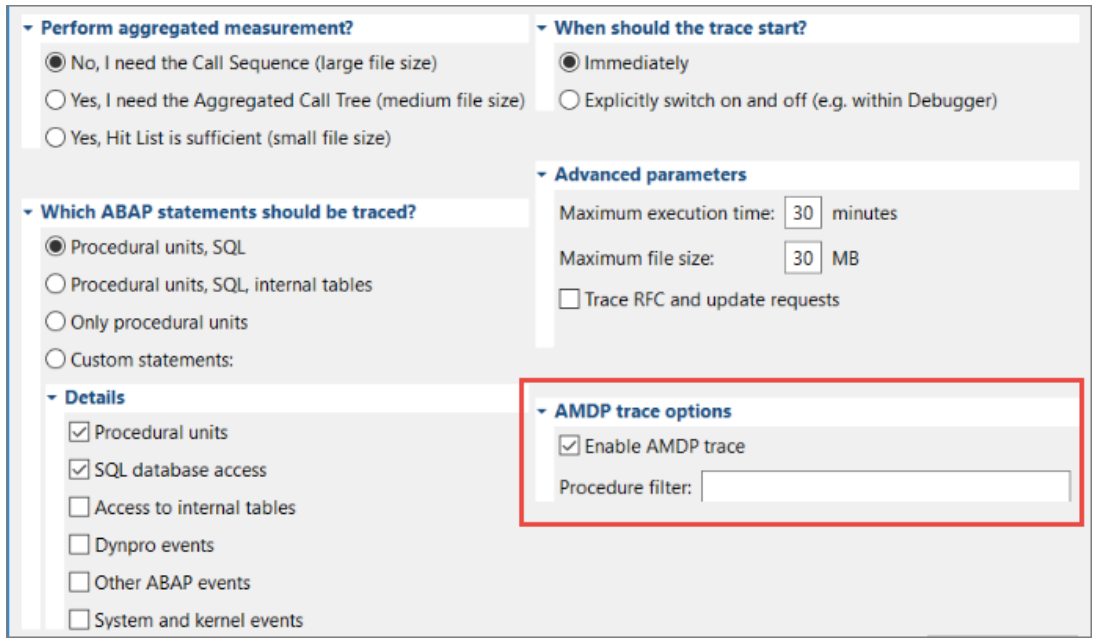
1. Switch to the ABAP Profiling perspective.
2. Open the ABAP Trace Requests view.
3. Choose your project and choose **Create Trace Request...** from the toolbar.



4. Choose the following options, then choose **Next**:
    - Which requests: Any
    - Object type: Any
    - Server: All servers
    - Maximum trace files... : 3
    - Title: Any for AMDP Demo



5. On the next screen, leave the other defaults, choose **Enable AMDP trace**, then choose **Finish**.



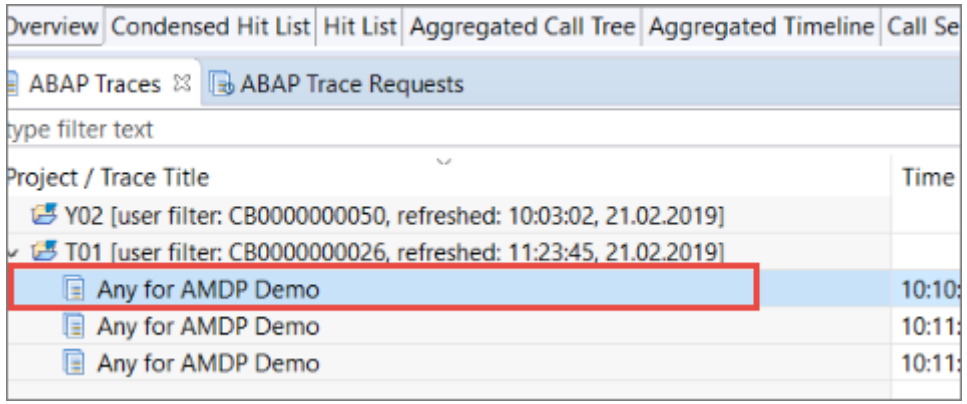
Done

Log on to answer question

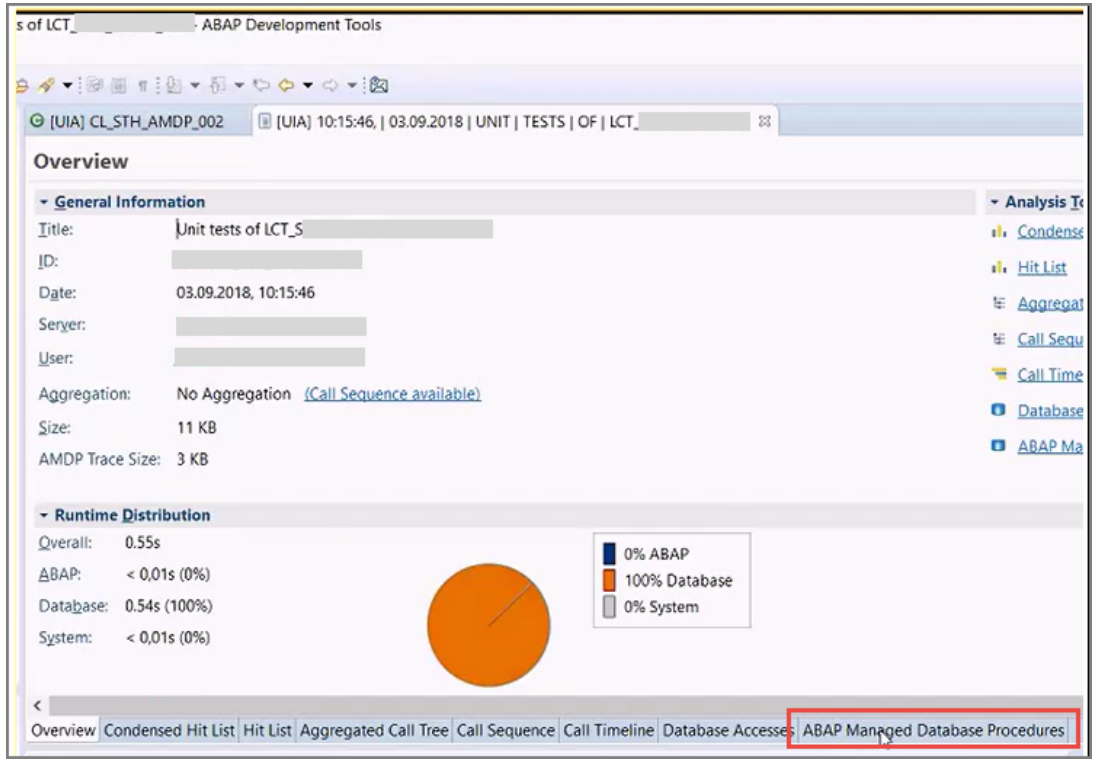
13 Step 13: Examine the AMDP Profiling Result

1. Refresh your trace request. Once it is finished, return to the ABAP Traces View.

2. Open the Profiling Overview by choosing (double-clicking) your trace.



3. Choose the ABAP Database Managed Procedures tab from the bottom.



4. The ABAP Managed Database Procedures overview displays the result of the AMDP profiling analysis in a table.

   To get more details about the statement which has been executed at ABAP runtime, view the table and expand the relevant procedure nodes.

5. Optional: To navigate to the relevant position within your ABAP source code, double-click the corresponding statement.The development object is then opened and the cursor is positioned at the relevant position.

You can now analyze the results of your AMDP profiling. For more details, see:- AMDP Profiling- Understanding AMDP Profiling Results

Done

Log on to answer question

14 Step 14: Test yourself

Define an AMDP method `get_sales_orders` (based on the AMDP method `get_flights`).Specify the options:

- Database = SAP HANA
- Language = SQLScript
- Options = read-only
- Objects used = `snwd_so, snwd_so_i`, and the method `read_sales_orders` of the class `zcl_demo`.

Do not indent your code.

Enter your code in the box below and choose **Submit Answer.**

Submit Answer

Log on to answer question ×