

How to Create a Dynamic ALV Report in SAP ABAP

This guide walks you through how to create a dynamic ALV report in SAP ABAP that can accept any table name at runtime, determine its structure, fetch data, and display the results in an ALV grid. By the end of this guide, you will have a flexible table viewer that works with any table in your SAP system.

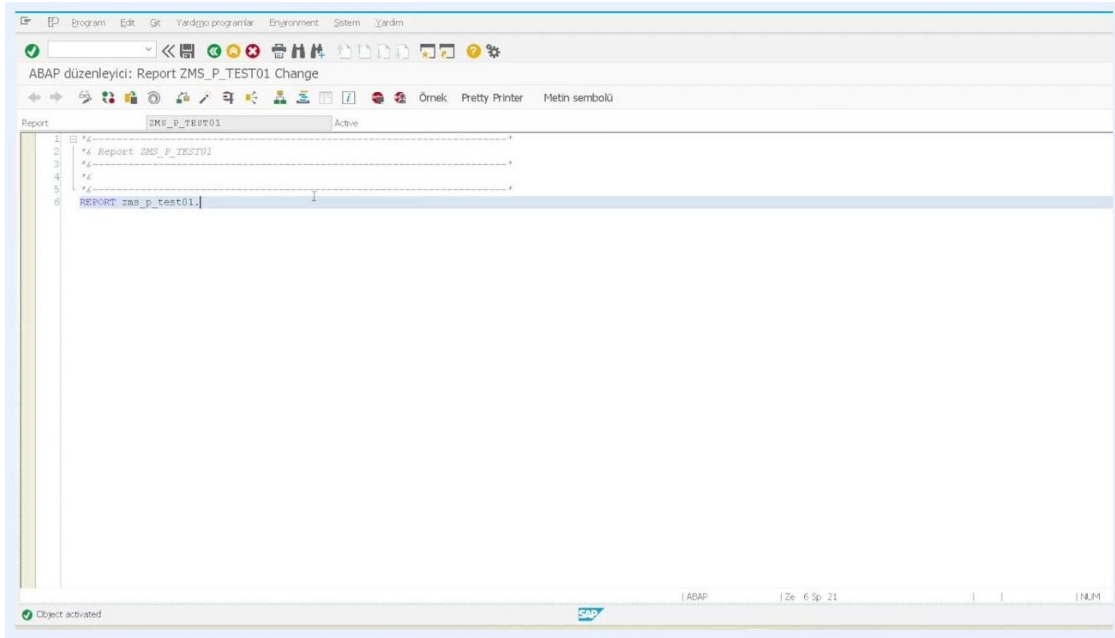
Prerequisites

- Access to the ABAP editor in your SAP system.
- Basic familiarity with ABAP syntax and the SAP data dictionary.

Step 1: Start a new ABAP report

Open the ABAP editor and create a new report named `zms_p_test01`. Enter the following declaration at the top of your report:

```
REPORT zms_p_test01.
```



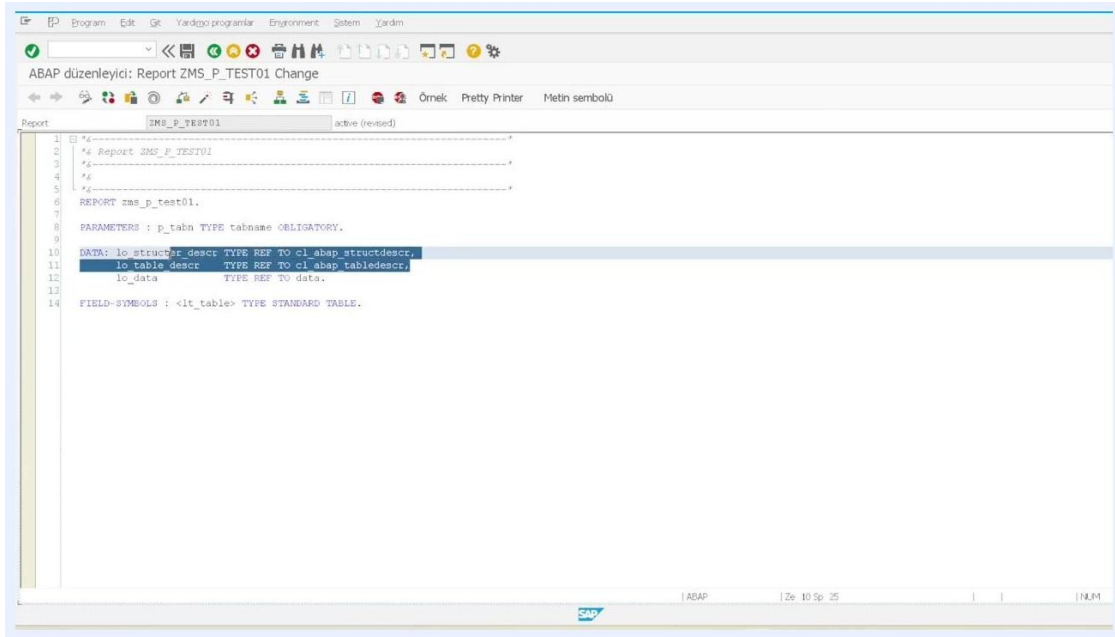
ABAP editor showing the initial report declaration: `REPORT zms_p_test01.`

Step 2: Define a parameter for table name input

Add a parameter to the selection screen so users can enter the name of the table they want to view. Define the parameter as obligatory and of type `tablename`:

```
PARAMETERS : p_tabn TYPE tablename OBLIGATORY.
```

This parameter accepts any table name, such as [echo](#) or [wubac](#).



Parameter definition for table name input: PARAMETERS : p_tabn TYPE tabname OBLIGATORY.

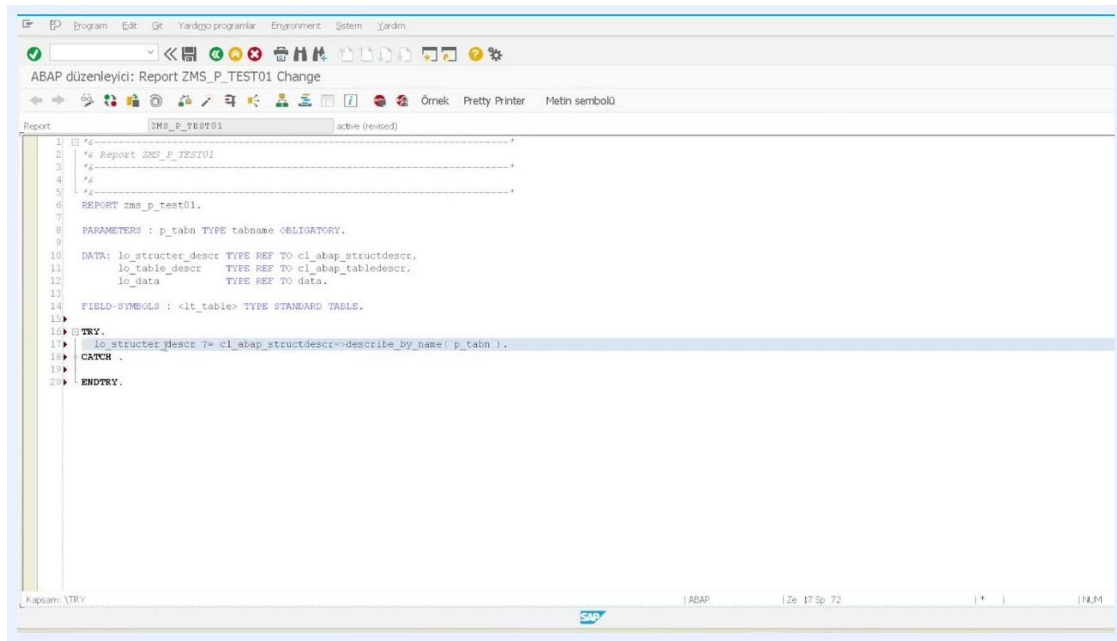
Step 3: Declare data objects for dynamic table handling

Define the data references needed to handle dynamic table structures at runtime:

- [lo_structur_descr](#) holds the structure definition.
- [lo_table_descr](#) holds the table definition.
- [lo_data](#) is the reference for the dynamic internal table.

You also need a field symbol to act as the dynamic internal table:

```
DATA: lo_structur_descr TYPE REF TO cl_abap_structdescr,  
      lo_table_descr   TYPE REF TO cl_abap_tabledescr,  
      lo_data          TYPE REF TO data.  
  
FIELD-SYMBOLS: <lt_table> TYPE STANDARD TABLE.
```



Data and field symbol declarations for dynamic table handling.

Step 4: Create the dynamic internal table

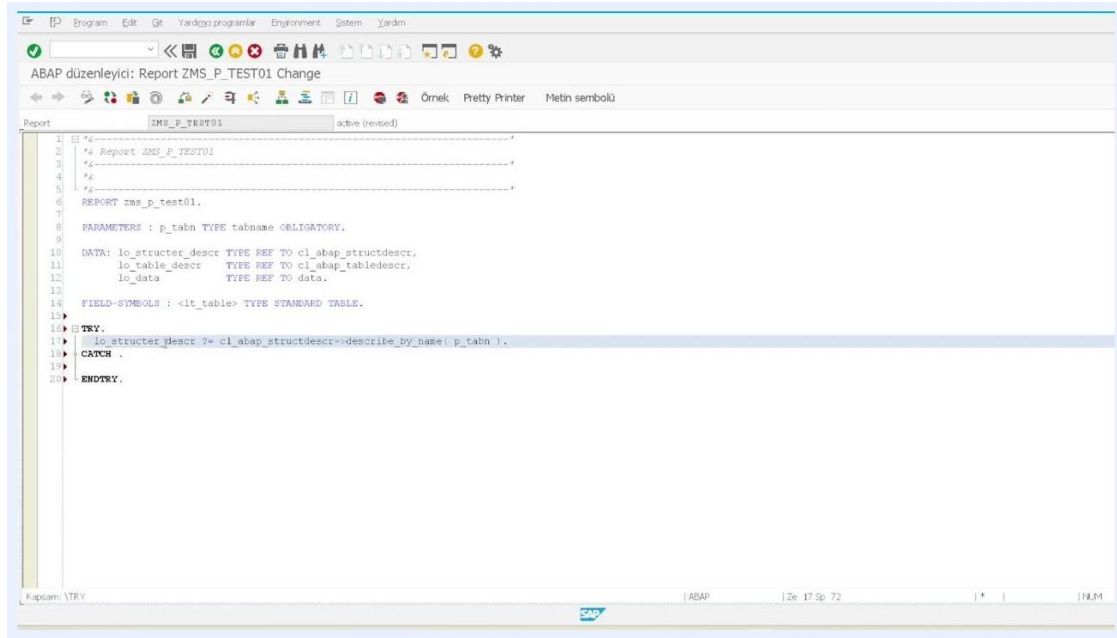
Determine the table structure at runtime

Use a **TRY-CATCH** block to safely determine the structure of the table entered by the user. Call `cl_abap_structdescr=>describe_by_name( p_tabn )` to retrieve the structure, then call `cl_abap_tabledescr=>create( lo_structer_descr )` to create a table description:

```

TRY.
    lo_structer_descr ?= cl_abap_structdescr=>describe_by_name( p_tabn
    ).
    lo_table_descr   =
cl_abap_tabledescr=>create( lo_structer_descr ).

```



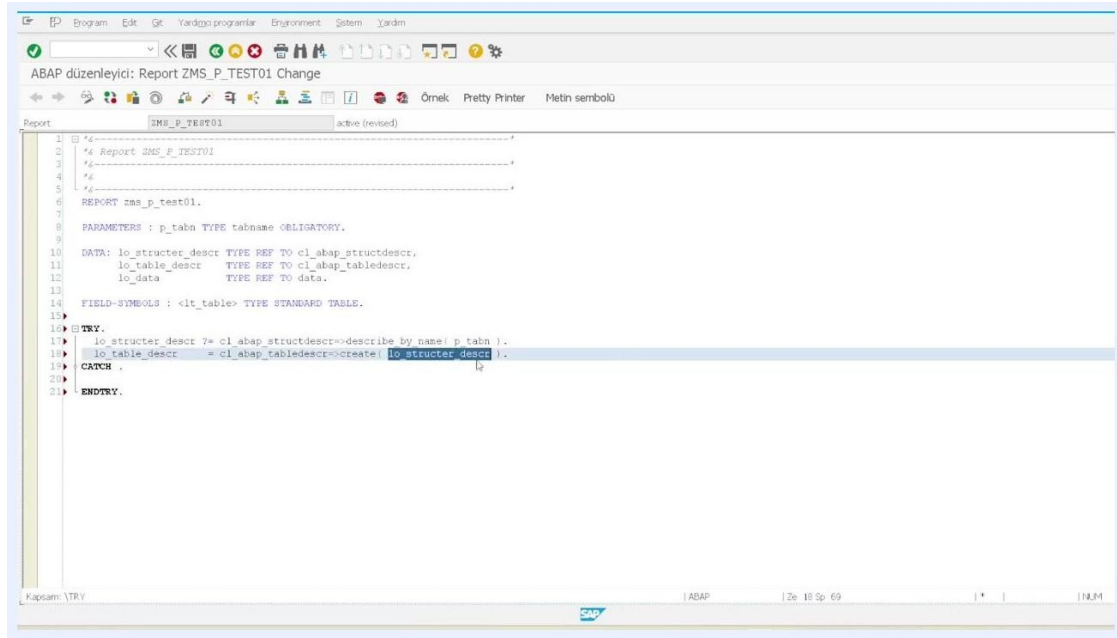
TRY block with structure and table description logic.

Create a data object based on the table template

Dynamically create a data object for the internal table using the table description handle, then assign that data object to the field symbol

`<lt_table>`:

```
CREATE DATA lo_data TYPE HANDLE lo_table_descr.  
ASSIGN lo_data->* TO <lt_table>.
```



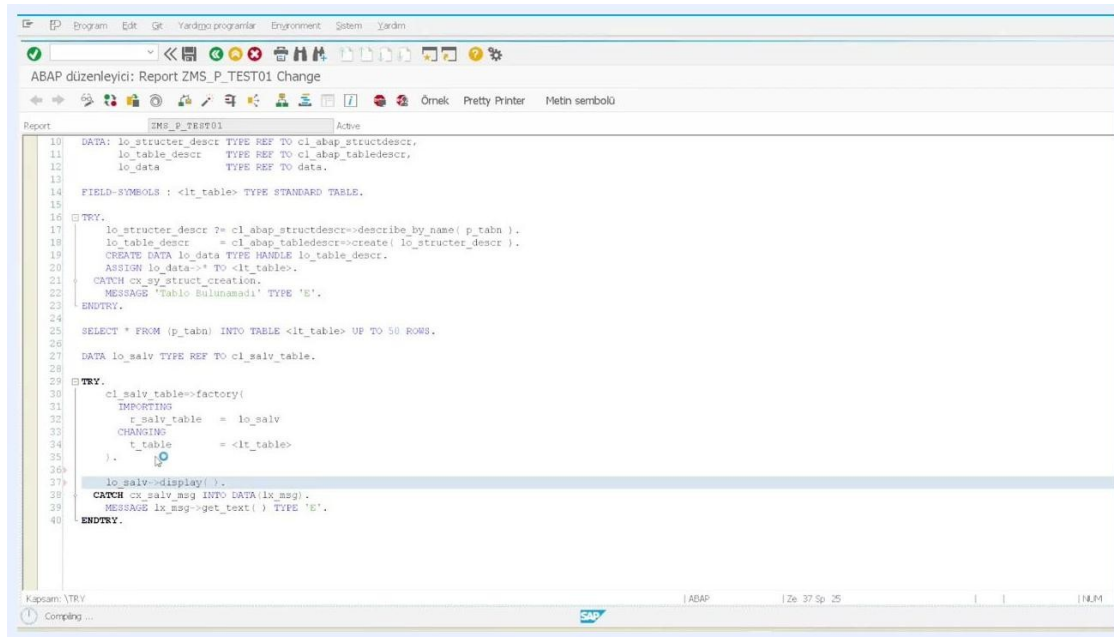
Data object creation and assignment to the field symbol.

Complete error handling for table structure creation

Add a **CATCH** block to handle cases where the entered table name does not exist, and display an error message if the structure cannot be created:

```
CATCH cx_sy_struct_creation.  
  MESSAGE 'Tablo Bulunamadı' TYPE 'E'.  
ENDTRY.
```

This ensures the program fails gracefully if the user enters an invalid table name.



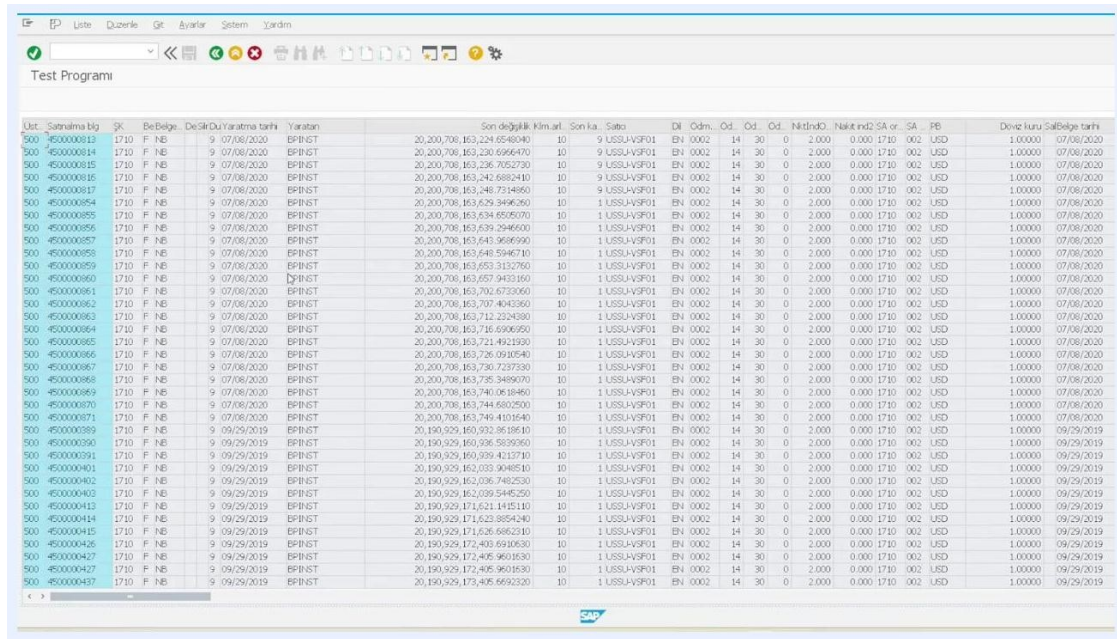
ABAP editor showing the completed TRY-CATCH-ENDTRY block for structure creation, with error message handling.

Step 5: Fetch the first 50 rows from the specified table

Use a dynamic SQL [SELECT](#) statement to retrieve up to 50 rows from the table specified by the user input ([p_tabn](#)). Store the results in the dynamically created internal table [<lt_table>](#):

```
SELECT * FROM (p_tabn) INTO TABLE <lt_table> UP TO 50 ROWS.
```

This step prepares the data for display.



ABAP editor showing the SELECT statement fetching up to 50 rows into the dynamic internal table.

Step 6: Prepare the ALV display

Declare a reference variable for the ALV table:

```
DATA lo_salv TYPE REF TO cl_salv_table.
```

Use a **TRY-CATCH** block to safely create and display the ALV table:

```
TRY.  
    cl_salv_table=>factory(  
        IMPORTING r_salv_table = lo_salv  
        CHANGING t_table      = <lt_table>  
    ).  
    lo_salv->display( ).  
CATCH cx_salv_msg INTO DATA(lx_msg).  
    MESSAGE lx_msg->get_text( ) TYPE 'E'.  
Generated with docsie.io
```

ENDTRY.

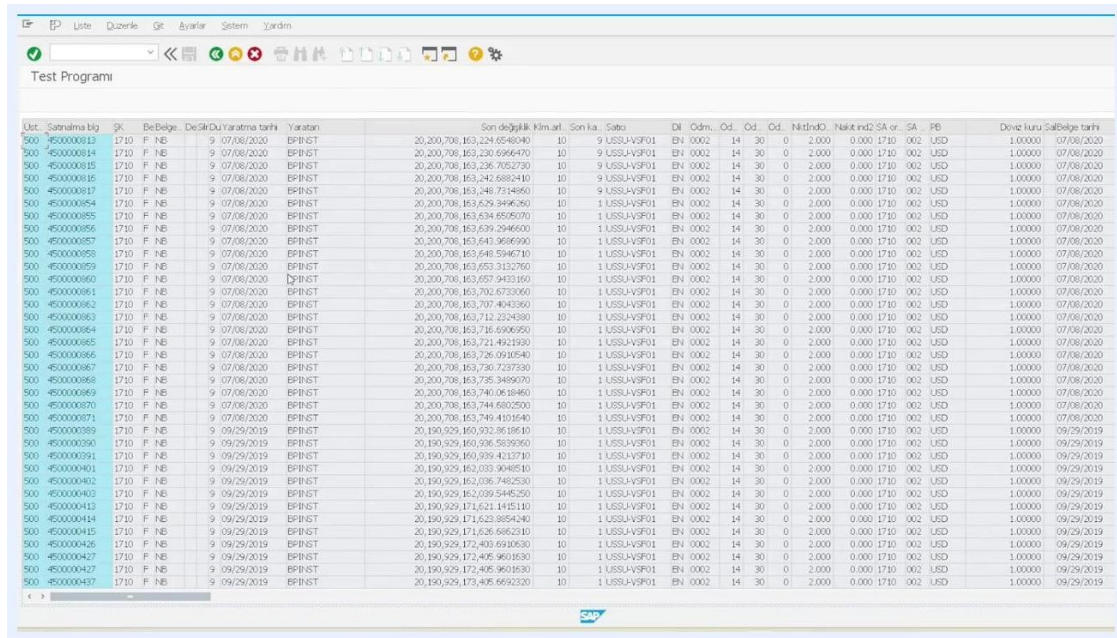
This code initializes the ALV viewer and displays the fetched data.

<

ABAP editor showing the ALV display logic with TRY-CATCH error handling.

Step 7: Run the program and view results

Activate and execute the report. Enter a valid table name (for example, [ECHO](#)) in the selection screen, then press **F8** to run the program. The first 50 rows from the specified table appear in an ALV grid.



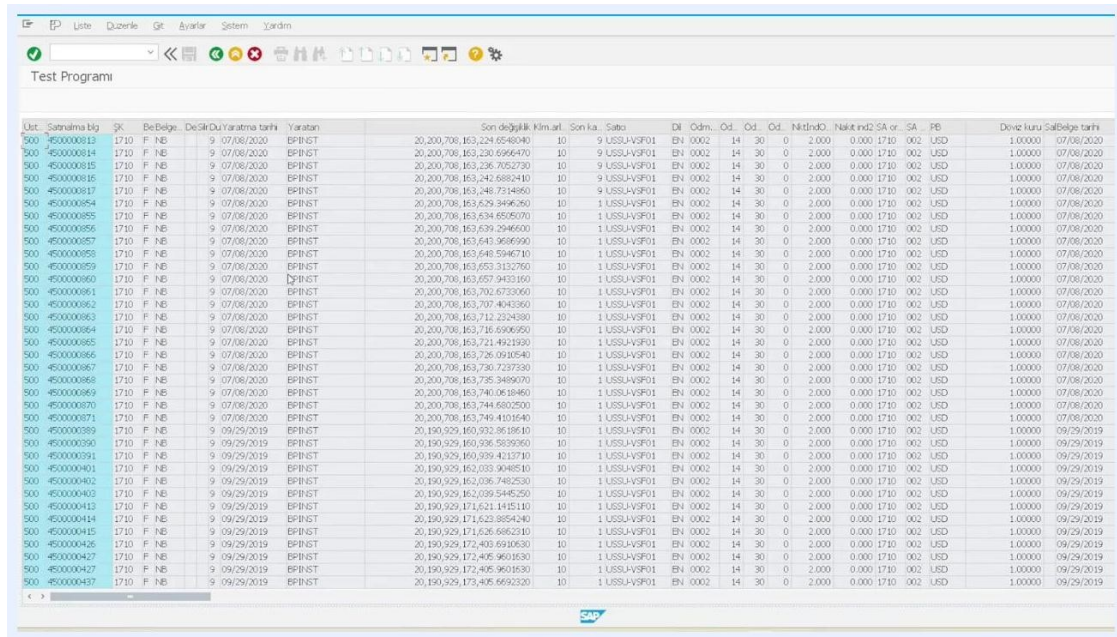
The screenshot shows the SAP Test Program interface. At the top, there's a menu bar with options like 'Liste', 'Düzenle', 'Gör', 'Ayarla', 'Sistem', and 'Yardım'. Below the menu is a toolbar with various icons. The main area is titled 'Test Program' and contains an ALV grid. The grid has 15 columns: 'Ust.', 'Sarımsa big', 'SK', 'Be Belge', 'De Sitr Du', 'Yaratma tarihi', 'Yaratari', 'Son değışlik', 'Kıml arı', 'Son ka', 'Sabo', 'Dİ', 'Odm.', 'Od.', 'Od.', 'Nıtlındı', 'Nakt ind2', 'SA or', 'SA', 'PB', and 'Dovz kuru'. The data rows show various values, including dates like '07/08/2020' and '09/29/2019', and numerical values like '20,200,706,163,224,654640'.

ALV grid displaying the first 50 rows of the selected table, with columns and data visible.

Step 8: Debug the dynamic table creation and data fetch

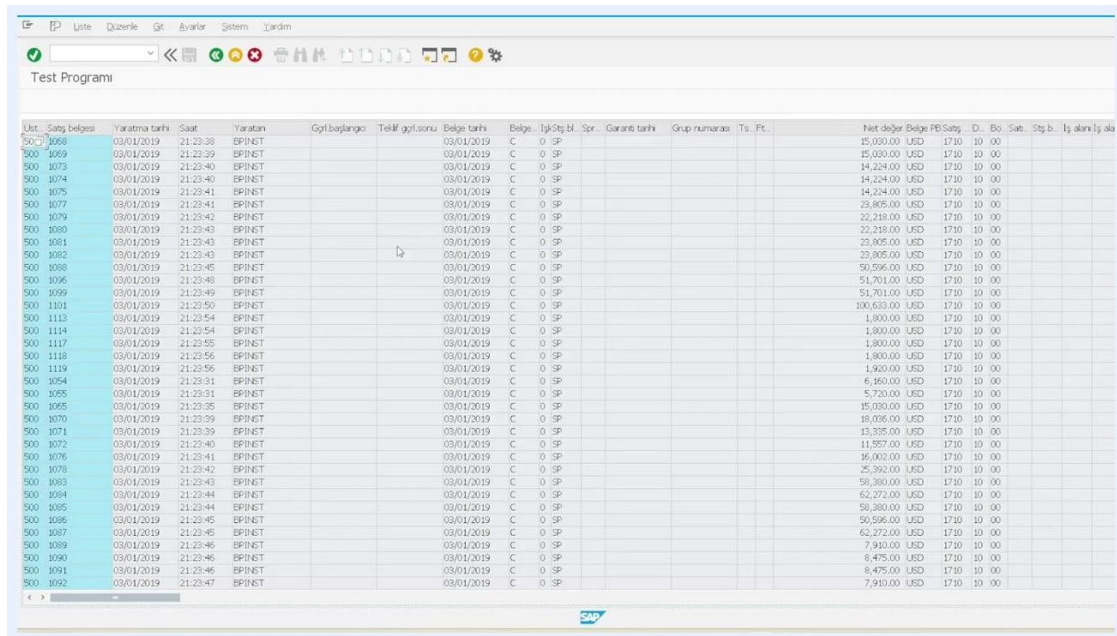
Set a breakpoint in the code to inspect the dynamic table creation and data fetching process, then run the program in debug mode. Observe the structure and contents of the dynamic internal table (`<lt_table>`) and the related references (`lo_structer_descr`, `lo_table_descr`).

Confirm that the structure is created from the table name, the table is created from the structure, and the data is fetched correctly.



The screenshot shows the SAP ABAP debugger interface. The top menu bar includes 'Liste', 'Düzenle', 'Gör', 'Ayarlar', 'Sistem', and 'Yardım'. Below the menu is a toolbar with various icons. The main window displays a table with the following columns: 'Ust.', 'Satınalma btlg', 'Gr.', 'Değer', 'Değer', 'Yararlan', 'Son değeri', 'Kısmi', 'Son', 'İla', 'Sato', 'DK', 'Cdm', 'Od', 'Od', 'Od', 'NidndK', 'NidndK2', 'SA', 'SA', 'PB', and 'Değer kuru'. The table contains 50 rows of data, with the first row starting with '500' and '4500000013'. The data is organized into a grid with alternating light and dark blue rows.

ABAP debugger showing the dynamic structure and table references, along with the internal table variable.

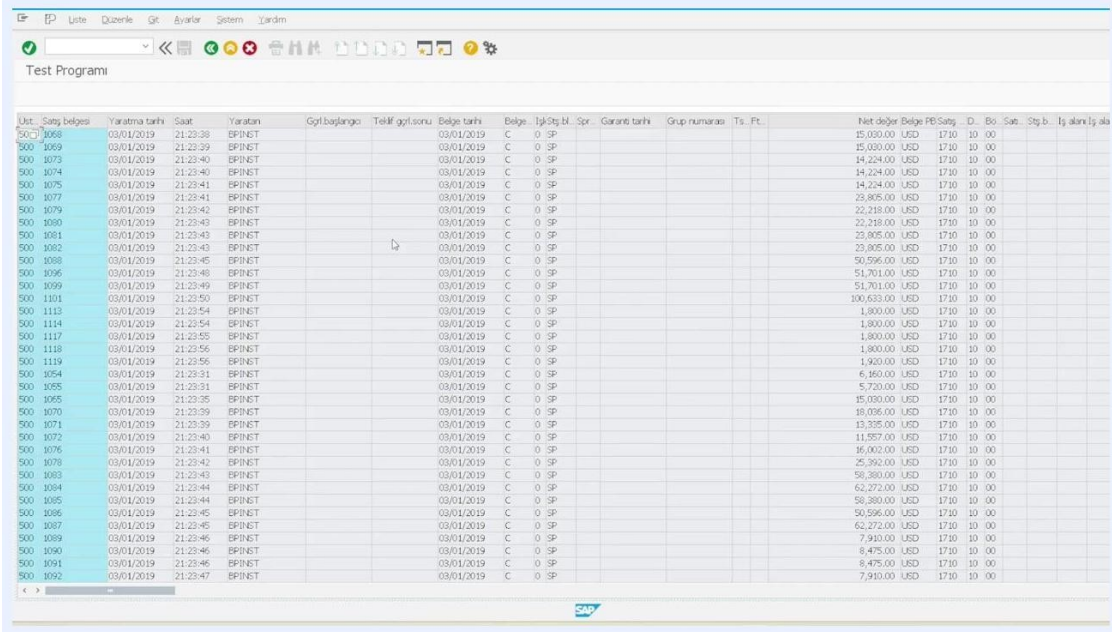


The screenshot shows the SAP ABAP debugger interface. The top menu bar includes 'Liste', 'Düzenle', 'Gör', 'Ayarlar', 'Sistem', and 'Yardım'. Below the menu is a toolbar with various icons. The main window displays a table with the following columns: 'Ust.', 'Satınalma btlg', 'Yararlan tarihi', 'Saat', 'Yararlan', 'Gör.baylango', 'Tetikl.gör.soru', 'Belge tarihi', 'Belge', 'İlgilı.btl', 'Spr.', 'Görantı tarihi', 'Grup numarası', 'Ts', 'Ft.', 'Net değeri', 'Belge PB Satış', 'D.', 'Bo.', 'Sat.', 'Stp.b.', 'İlgilı.btl', and 'İlgilı.btl'. The table contains 50 rows of data, with the first row starting with '500' and '4500000013'. The data is organized into a grid with alternating light and dark blue rows.

ABAP debugger showing the internal table <lt_table> filled with 50 rows, along with the table name parameter.

Step 9: Verify ALV output for different tables

Test the program with different table names (for example, [VUBAC](#)). Each time, the ALV grid displays the first 50 rows of the specified table, confirming the dynamic nature of the solution.



Doc. Satış belgesi	Yaratma tarihi	Saat	Yaratan	Grl.baslangici	Teklif gyl.usuru	Belge tarihi	Belge	lgkStp.bl	Spr.	Garanti tarihi	Grup numarası	Ts.	Pt.	Net değer	Belge PB	Satış	D.	Bo.	Sat.	Stp.b.	İlg. alanı	İlg. ala
500 1068	03/01/2019	21:23:38	BPIN6T			03/01/2019	C	0	SP					25,030.00	USD	1710	10	00				
500 1069	03/01/2019	21:23:39	BPIN6T			03/01/2019	C	0	SP					25,030.00	USD	1710	10	00				
500 1073	03/01/2019	21:23:40	BPIN6T			03/01/2019	C	0	SP					14,224.00	USD	1710	10	00				
500 1074	03/01/2019	21:23:40	BPIN6T			03/01/2019	C	0	SP					14,224.00	USD	1710	10	00				
500 1075	03/01/2019	21:23:41	BPIN6T			03/01/2019	C	0	SP					14,224.00	USD	1710	10	00				
500 1077	03/01/2019	21:23:41	BPIN6T			03/01/2019	C	0	SP					23,805.00	USD	1710	10	00				
500 1079	03/01/2019	21:23:42	BPIN6T			03/01/2019	C	0	SP					22,218.00	USD	1710	10	00				
500 1090	03/01/2019	21:23:43	BPIN6T			03/01/2019	C	0	SP					22,218.00	USD	1710	10	00				
500 1081	03/01/2019	21:23:43	BPIN6T			03/01/2019	C	0	SP					23,805.00	USD	1710	10	00				
500 1082	03/01/2019	21:23:43	BPIN6T			03/01/2019	C	0	SP					23,805.00	USD	1710	10	00				
500 1088	03/01/2019	21:23:45	BPIN6T			03/01/2019	C	0	SP					50,596.00	USD	1710	10	00				
500 1096	03/01/2019	21:23:46	BPIN6T			03/01/2019	C	0	SP					51,701.00	USD	1710	10	00				
500 1099	03/01/2019	21:23:49	BPIN6T			03/01/2019	C	0	SP					51,701.00	USD	1710	10	00				
500 1101	03/01/2019	21:23:50	BPIN6T			03/01/2019	C	0	SP					100,633.00	USD	1710	10	00				
500 1113	03/01/2019	21:23:54	BPIN6T			03/01/2019	C	0	SP					1,800.00	USD	1710	10	00				
500 1114	03/01/2019	21:23:54	BPIN6T			03/01/2019	C	0	SP					1,800.00	USD	1710	10	00				
500 1117	03/01/2019	21:23:55	BPIN6T			03/01/2019	C	0	SP					1,800.00	USD	1710	10	00				
500 1118	03/01/2019	21:23:56	BPIN6T			03/01/2019	C	0	SP					1,800.00	USD	1710	10	00				
500 1119	03/01/2019	21:23:56	BPIN6T			03/01/2019	C	0	SP					1,920.00	USD	1710	10	00				
500 1054	03/01/2019	21:23:31	BPIN6T			03/01/2019	C	0	SP					6,150.00	USD	1710	10	00				
500 1055	03/01/2019	21:23:31	BPIN6T			03/01/2019	C	0	SP					5,720.00	USD	1710	10	00				
500 1095	03/01/2019	21:23:35	BPIN6T			03/01/2019	C	0	SP					15,030.00	USD	1710	10	00				
500 1070	03/01/2019	21:23:39	BPIN6T			03/01/2019	C	0	SP					18,036.00	USD	1710	10	00				
500 1071	03/01/2019	21:23:39	BPIN6T			03/01/2019	C	0	SP					13,335.00	USD	1710	10	00				
500 1072	03/01/2019	21:23:40	BPIN6T			03/01/2019	C	0	SP					11,557.00	USD	1710	10	00				
500 1076	03/01/2019	21:23:41	BPIN6T			03/01/2019	C	0	SP					16,002.00	USD	1710	10	00				
500 1078	03/01/2019	21:23:42	BPIN6T			03/01/2019	C	0	SP					25,392.00	USD	1710	10	00				
500 1083	03/01/2019	21:23:43	BPIN6T			03/01/2019	C	0	SP					58,380.00	USD	1710	10	00				
500 1084	03/01/2019	21:23:44	BPIN6T			03/01/2019	C	0	SP					62,272.00	USD	1710	10	00				
500 1085	03/01/2019	21:23:44	BPIN6T			03/01/2019	C	0	SP					58,380.00	USD	1710	10	00				
500 1086	03/01/2019	21:23:45	BPIN6T			03/01/2019	C	0	SP					50,596.00	USD	1710	10	00				
500 1087	03/01/2019	21:23:45	BPIN6T			03/01/2019	C	0	SP					62,272.00	USD	1710	10	00				
500 1089	03/01/2019	21:23:46	BPIN6T			03/01/2019	C	0	SP					7,910.00	USD	1710	10	00				
500 1090	03/01/2019	21:23:46	BPIN6T			03/01/2019	C	0	SP					8,475.00	USD	1710	10	00				
500 1091	03/01/2019	21:23:46	BPIN6T			03/01/2019	C	0	SP					8,475.00	USD	1710	10	00				
500 1092	03/01/2019	21:23:47	BPIN6T			03/01/2019	C	0	SP					7,910.00	USD	1710	10	00				

ALV grid displaying the first 50 rows of a different table, with columns and data visible.

Summary

By following these steps, you have learned how to create a dynamic ALV report in SAP ABAP that:

- Accepts any table name at runtime.
- Determines its structure automatically.
- Fetches the first 50 rows of data.
- Displays the results in an ALV grid.
- Handles errors gracefully.
- Supports debugging for verification.

You can now use this report as a flexible table viewer for any table in your SAP system.