SAP ABAP New Syntax

CONTENTS

1. INLINE DATA DECLARATIONS
2. NEW FEATURES OF OPEN SQL
3. CONDITIONAL STATEMENT
4. String Expression as a Replacement of CONCATENATE
5. VALUE EXPRESSION
6. CORRESPONDING OPERATOR
7. ALPHA KEYWORD
8. LET EXPRESSION
9. REDUCE OPERATOR
10. FILTER OPERATOR
11. TABLE EXPRESSION
12. Group By Operator
13. Object Oriented

1) Inline Data Declarations

- In the new way of data declaration, there is no need to declare the objects separately.
- We can declare the object where we use it.
- The keyword **DATA** is used for inline data declarations.

New Syntax:

- ➢ Declaring variables and assigning values to it at the same instance.

```
DATA(lv_input1) = 10.
DATA(lv_input2) = 20.
DATA(lv_output) = lv_input1 + lv_input2.
```

- ➢ Declaration of the work area while operation on Internal Table

```
LOOP AT lt_data INTO DATA(lwa_data).
  WRITE: / lwa_data-vbeln, lwa_data-erdat, lwa_data-erzet, lwa_data-
ernam, lwa_data-description.
ENDLOOP.
```

- ➢ Assigning Field Symbols.

```
LOOP AT lt_data ASSIGNING FIELD-SYMBOL(<fs_vbak>).
  " Write the fields of each record to the output
  WRITE: / <fs_vbak>-vbeln, <fs_vbak>-erdat, <fs_vbak>-ernam, <fs_vbak>-
netwr, <fs_vbak>-waerk.
ENDLOOP.
UNASSIGN <fs_vbak>.
```

2) New Features of OPEN SQL

- The separator between the columns is comma (,).
- ABAP data objects used in open SQL statements are called **host variables** and they are prefixed with **'@'**.

New Syntax:

- ➢ Selecting Data from Database Table into an Internal Table

```
SELECT vbeln,erdat,erzet,ernam,vbtyp
FROM vbak
INTO TABLE @lt_data
WHERE vbeln IN @s_vbeln.
```

- ➢ We can use conditional statements like CASE in the select list.

```
SELECT vbeln, erdat, erzet, ernam, vbtyp,
CASE vbtyp
WHEN 'C' THEN 'Internal Order'
ELSE 'External Order'
END AS description
FROM vbak
INTO TABLE @DATA(lt_data)
WHERE vbeln IN @s_vbeln
ORDER BY vbeln DESCENDING.
```

- ➢ We can use the literals in the select list.

```
SELECT SINGLE 'X'
FROM z24731_user
INTO @DATA(lv_exists)
WHERE uname = @sy-uname.
```

- ➢ We can use the aggregate functions like SUM, COUNT, MAX, MIN, AVG & we can pass the element list after FROM clause also. When we specify the element list after FROM clause, we need to use FIELDS keyword.

```
SELECT  FROM vbak
FIELDS erdat,SUM( netwr ) AS netwr
WHERE erdat IN @s_erdat
GROUP BY erdat,vbtyp HAVING SUM( netwr ) > @p_netwr
INTO TABLE @DATA(lt_data).
```

- ➢ We can pass the element list after FROM clause also. When we specify the element list after FROM clause, we need to use FIELDS keyword.

```
SELECT SINGLE seatsmax - seatsocc AS lv_seats_e,
              seatsmax_b - seatsocc_b AS lv_seats_b
FROM sflight
INTO @DATA(lwa_data)
WHERE carrid = @p_carrid
AND connid = @p_connid
AND fldate = @p_fldate.
```

3) Conditional Statement

- In SAP ABAP new syntax, instead of IF statement, we can use the COND statement.
- In SAP ABAP new syntax, instead of IF statement, we can use the COND statement.

New Syntax:

- ➢ COND statement as replacement of IF

```
lv_text = COND #( WHEN lwa_data-netwr GE 0 AND lwa_data-netwr LE 5000
                  THEN TEXT-000
                  WHEN lwa_data-netwr > 5000 AND lwa_datnetwr LE 10000
                  THEN TEXT-001
                  ELSE TEXT-002 ).
```

- ➢ SWITCH statement as replacement of CASE

```
lv_switch = SWITCH #( p_dept WHEN 'HR' THEN 'Human Resources' WHEN 'Finance' THEN
 'Finance Department'
              ELSE 'Other Department' ).
```

4) String Expression as a Replacement of CONCATENATE

- In SAP ABAP new syntax, instead of CONCATENATE, we can use the string expressions.
  While using string expression, we need to use the pipe(|) symbol at the starting and at the end.
  The variables must be enclosed in curly brackets{}.

New Syntax:

- Concatenate using ‘{}’ operator.

```
      DATA(lv_input1) = 'welcome'.
      DATA(lv_input2) = 'to'.
      DATA(lv_input3) = 'home'.


DATA(lv_output) = |{ lv_input1 } { lv_input2 } { lv_input3 }|.
```

- Concatenate using ‘&&’ operator.

```
DATA(lv_concat_amp) = 'I have ' && ls_vbak-netwr && ' Rupees'.
ULINE.
WRITE :/ 'Concatenate using &&', lv_concat_amp.
```

5) VALUE Expression

- In SAP ABAP new syntax, instead of APPEND statement, we can use VALUE expression.
- When defining a standard table type in SAP ABAP, it is necessary to specify the key or write with empty key.
- When we don’t want to overwrite the existing values of the internal table we use **BASE** in value expression.
  The values of the BASE table acts as a initial values for the target table and after that the new values are inserted to the target table.
  It is very useful in loops when we want to append the rows to the target internal table using VALUE expression.
- When we use ‘#’ character for type and we use it with value keyword we cannot do the inline data declaration. Internal tables or work areas should be explicitly defined.

New Syntax:

- Creating an Internal Table and Filling Data into it.

```
TYPES: BEGIN OF lty_data,
  eid(10) TYPE N,
  ename(40) TYPE c,
  END OF lty_data.

TYPES: ltty_data TYPE TABLE of lty_data WITH EMPTY KEY.

 "DATA(lwa_data) = VALUE lty_data( eid = 1 ename = 'AYAN' )
 DATA(lt_data) = VALUE ltty_data( ( eid = 1 ename = 'Nabamit' ) ( eid = 2 ename =
 'Suvodip' ) ( eid = 3 ename = 'Navatej' ) ).

 LOOP AT lt_data INTO DATA(lwa_data1).
   WRITE: / lwa_data1-eid , lwa_data1-ename.
 ENDLOOP.
```

- Adding New entries to Internal Table already having entries

```
lt_data = VALUE #( BASE lt_data( eid = 4 ename = 'SRUJAN' ) ( eid = 5 ename = 'SH
IVAM' ) ( eid = 6 ename = 'NIVEDHA' ) ).
```

6) CORRESPONDING Operator

- In SAP ABAP new syntax, The CORRESPONDING operator is used to move data between the internal tables.
- CORRESPONDING - Moves data between the internal tables for the matching columns.
- CORRESPONDING WITH MAPPING - This is used to map the data of one column to another column.
- CORRESPONDING WITH EXCEPT - This is used to move the data of the selected matching columns.

NEW SYNTAX:

```
TYPES: BEGIN OF lty_data,
         vbeln TYPE vbeln_va,
         erdat TYPE erdat,
         erzet TYPE erzet,
         ernam TYPE ernam,
         vbtyp TYPE vbtyp,
       END OF lty_data.
TYPES: BEGIN OF lty_data1,
         vbeln TYPE vbeln_va,
         uname TYPE ernam,
         vbtyp TYPE vbtyp,
       END OF lty_data1.
DATA :lt_data TYPE TABLE OF lty_data.
DATA :lwa_data TYPE lty_data.
DATA :lt_data1 TYPE TABLE OF lty_data1.
DATA :lwa_data1 TYPE  lty_data1.

SELECT vbeln,erdat,erzet,ernam,vbtyp
FROM vbak
INTO TABLE @lt_data
WHERE vbeln IN @s_vbeln.

lt_data1 = CORRESPONDING #( lt_data )." only the data for the matching field name
 in both the table is is moved.
  lt_data1 = CORRESPONDING #( lt_data MAPPING uname = ernam  EXCEPT vbtyp ).
*lt_data1 = CORRESPONDING #( lt_data MAPPING uname = ernam ).
```

7) ALPHA Keyword with IN or OUT as a Replacement of Conversion FM's

- In SAP ABAP new syntax, instead of using these function modules, we can use the ALPHA keyword with IN which is used to add the leading zero's and ALPHA keyword with OUT to remove the leading zero's.

New Syntax :

```
DATA : lv_input1(10) TYPE c VALUE '12345'.
DATA : lv_output1(10) TYPE c.
DATA : lv_input2(10) TYPE c VALUE '0000019999'.
DATA : lv_output2(10) TYPE c.
lv_output1 = |{ lv_input1 ALPHA = in }|.
lv_output2 = |{ lv_input2 ALPHA = out }|.

WRITE: / lv_output1,
       / lv_output2.
```

8) LET EXPRESSION

- A LET expression defines variables or field symbols in an expression and assign values to them.

New Syntax:

```
TYPES: BEGIN OF lty_data,
         col1(2) TYPE n,
         col2(3) TYPE n,
       END OF lty_data.
DATA : lwa_data TYPE lty_data.
*DATA(x) = 10.
DO p_input TIMES.
  lwa_data = VALUE #( LET x = 10 IN col1 = sy-index col2 = sy-index + x ).
  WRITE : / lwa_data-col1, lwa_data-col2.
ENDDO.
```

9) REDUCE Operator

- In SAP ABAP new syntax, instead of LOOP, we can use REDUCE operator to calculate the count , total, subtotal etc.
- The syntax to use a REDUCE operator is as follows:
- REDUCE <type>(INIT ... FOR... THEN... UNTIL... NEXT...)
- In the syntax - REDUCE = keyword, <type> = data type of the result , INIT = initial value , FOR = initialize iteration , THEN = increment , UNTIL = iteration end condition , NEXT = operation.

New Syntax:

```
* Calculate the number of sales orders for each document category using REDUCE
DATA(lv_count1) = REDUCE i( INIT count = 0 FOR lwa_data IN lt_data
                            WHERE ( vbtyp = 'A' ) NEXT count = count + 1 ).
WRITE : 'The number of sales orders for document category A:', lv_count1.
```

10) FILTER OPERATOR

- It is used to filter the internal table and get the subset of data into a new internal table.
- There are 2 ways to apply the FILTER operator.
- FILTER with single values - In this we pass the filtering values using WHERE condition and filtered data appears into new internal table.
- FILTER with filter table - In this we need two internal tables. One internal table has the actual data on which filtering is applied and another is filter internal table which has the filter values used for filtering.

New Syntax:

```
TYPES: BEGIN OF lty_data,
         vbeln TYPE vbeln_va,
         vbtyp TYPE vbtypl,
         erzet TYPE erzet,
         ernam TYPE ernam,
         erdat TYPE erdat,
       END OF lty_data.
TYPES: BEGIN OF lty_filter,
         vbtyp TYPE vbtypl,
       END OF lty_filter,
DATA: lt_temp_data TYPE TABLE OF lty_data.
DATA: lt_filter TYPE SORTED TABLE OF lty_filter WITH UNIQUE KEY vbtyp.
DATA: lt_data TYPE SORTED TABLE OF lty_data WITH NON-UNIQUE KEY vbtyp.

SELECT vbeln, vbtyp, erzet, ernam, erdat
  FROM vbak
  INTO TABLE @lt_data
  WHERE erdat IN @s_erdat.
```

➢ Applying filter operator using single values.

```
lt_temp_data = FILTER #( lt_data  WHERE vbtyp = CONV vbtypl('A') ).
```

➢ Applying Filter operator using filter table.

```
lt_filter = VALUE #( ( vbtyp ='A' ) ( vbtyp ='C' ) ).
*lt_temp_data = FILTER #( lt_data  IN lt_filter WHERE vbtyp = vbtyp ).
lt_temp_data = FILTER #( lt_data  EXCEPT IN lt_filter WHERE vbtyp = vbtyp ).
```

11) Table Expressions as Replacement of READ TABLE

- In SAP ABAP new syntax, instead of READ TABLE, we can use the table expressions.
- We need to use the square bracket[ ].
- In the square bracket, we need to specify the key or index.

New Syntax:

➢ Get the line from internal table to work area using index

```
DATA(lwa_data) = lt_data[ 10 ].
```

➢ Get the line from internal table to work area using key

```
DATA(lwa_data) = lt_data[ ernam = 'TRAINEE1841' ].
```

`Important: sy-subrc doesn’t work in table expression`

- While using table expression, If the table entry does not exist, system throws the exception CX_SY_ITAB_LINE_NOT_FOUND.
- We can use **line_exists** statement or **TRY/CATCH block** or **VALUE expression with OPTIONAL** keyword to check the existence of a record.

Syntax

➢ Using Line exists to check if the record is present in the internal table or not.

```
IF line_exists( lt_data[ 10 ] ).
  DATA(lwa_data) = lt_data[ 10 ].
  WRITE : / lwa_data-vbeln , lwa_data-erdat, lwa_data-erzet, lwa_data-ernam.
ELSE.
  WRITE:/ TEXT-000.
ENDIF.
```

- ➢ Using Try/catch block for checking existence of record.

```
TRY.
    DATA(lwa_data) = lt_data[ 10 ].
  CATCH
    cx_sy_itab_line_not_found INTO lo_object.
    CALL METHOD lo_object->if_message~get_text
      RECEIVING
        result = lv_result.
ENDTRY.

IF lwa_data IS NOT INITIAL.
  WRITE : / lwa_data-vbeln , lwa_data-erdat, lwa_data-erzet,lwa_data-ernam.
ELSE.
  WRITE : / lv_result.
ENDIF.
```

- ➢ Using VALUE expression with OPTIONAL keyword

```
DATA(lwa_data) = VALUE #( lt_data[ 10 ] OPTIONAL ).
```

12) Group BY Operator

- The GROUP BY clause is used to group the records

New Syntax :

- ➢ Group By operations instead of multiple loops.

```
SORT gt_eban BY ekgrp matnr.
  LOOP AT gt_eban INTO DATA(lw_eban)
    GROUP BY ( ekgrp = lw_eban-ekgrp )
*     ASCENDING
*     REFERENCE INTO DATA(lr_eban).
    INTO DATA(lr_eban).
    WRITE :/ 'start of the group'.
    LOOP AT GROUP lr_eban INTO DATA(lw_print).
      WRITE:/ lw_print-banfn,lw_print-matnr,lw_print-menge.
    ENDLOOP.
  ENDLOOP.
```

13) Object Oriented

- In SAP ABAP new syntax, instead of CREATE OBJECT, we can use the NEW keyword to create an object of the class.

New Syntax:

- ➢ Creating an instance of a class.

```
DATA(lo_instance) = NEW lcl_simple( ).
```

- ➢ Calling a static class

```
lt_bin = cl_document_bcs=>xstring_to_solix( ip_xstring = xl_content )
```