Capgemini

SAP HANA

Lesson Name: ABAP New syntax

Contents

- Inline data declaration
- Explicit type declaration
- Standard internal table declaration
- Sorted internal table declaration
- Internal table with more components
- How to work with Deep structure
- MOVE-CORRESPONDING for Internal Tables
- Table expressions
- GROUP BY for Internal Tables
- FILTER expressions
- INNER JOIN
- NEW keyword for creating Objects
- CONVERSION_EXIT_ALPHA_INPUT/OURPUT
- Using SWITCH statement

Lesson Objectives

After completing this lesson, participants will be able to -

- Know ABAP New syntax (SAP NW 7.4 onwards)
- Being fluent to the basic up gradations of coding in SAP
- Learning new SAP provided facilities from ABAP 7.4
- Adapting with the new syntaxes form 7.4
- Log on to SAP and do the Basic Navigations

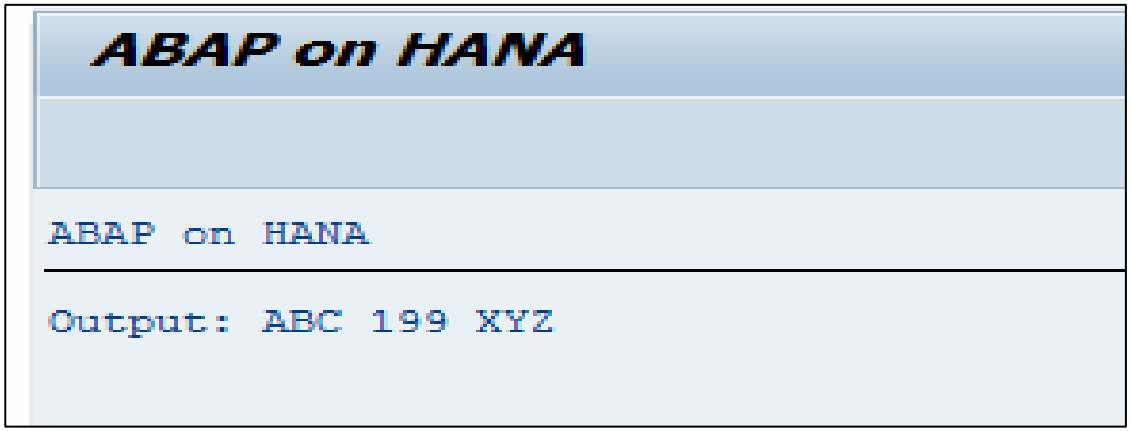
Inline data declaration

Definition: Inline data declarations are a new way of declaring variables and field symbols at operand positions

Declaration of Variable:

```
DATA(V_DATA) =  'ABC 199 XYZ'.
WRITE: 'Output:', V_DATA.
```

Output:



Inline data declaration

Declaration of table work areas:

```
LOOP AT itab INTO DATA(wa).
  ...
ENDLOOP.
```

Declaration of actual parameters:

Old method

```
DATA a1 TYPE …
DATA a2 TYPE …
oref->meth( IMPORTING p1 = a1
            IMPORTING p2 = a2
           … )
```

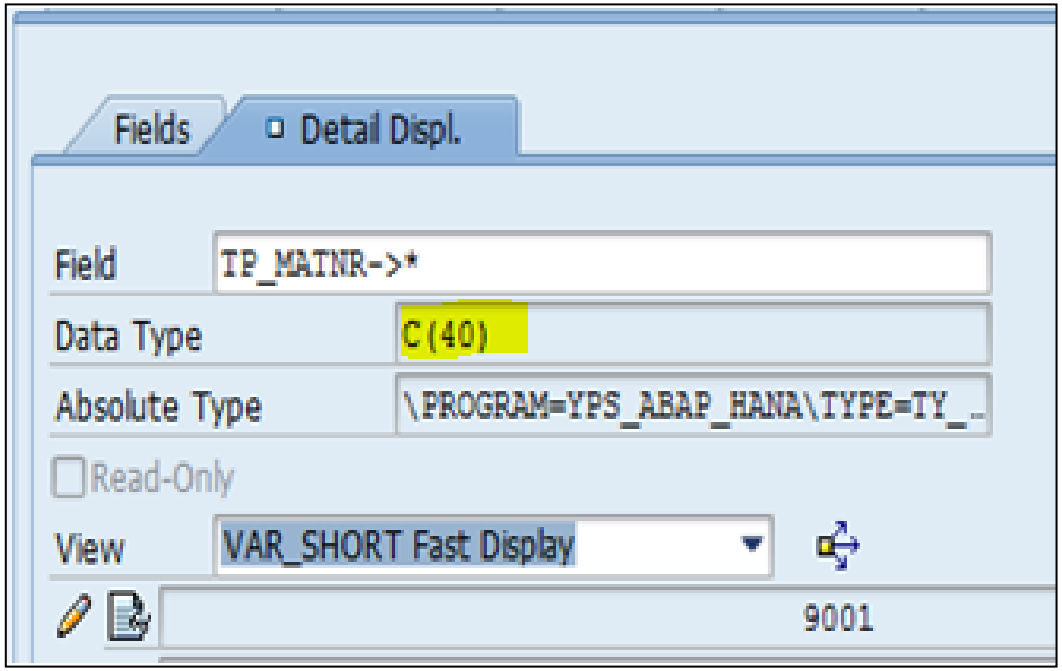
New Method

```
oref->meth( IMPORTING p1 = DATA(a1)
            IMPORTING p2 = DATA(a2)
           … ).
```

Explicit type declaration

```
TYPES ty_matnr TYPE matnr.
DATA(tp_matnr) = NEW ty_matnr( 9001 ).
```

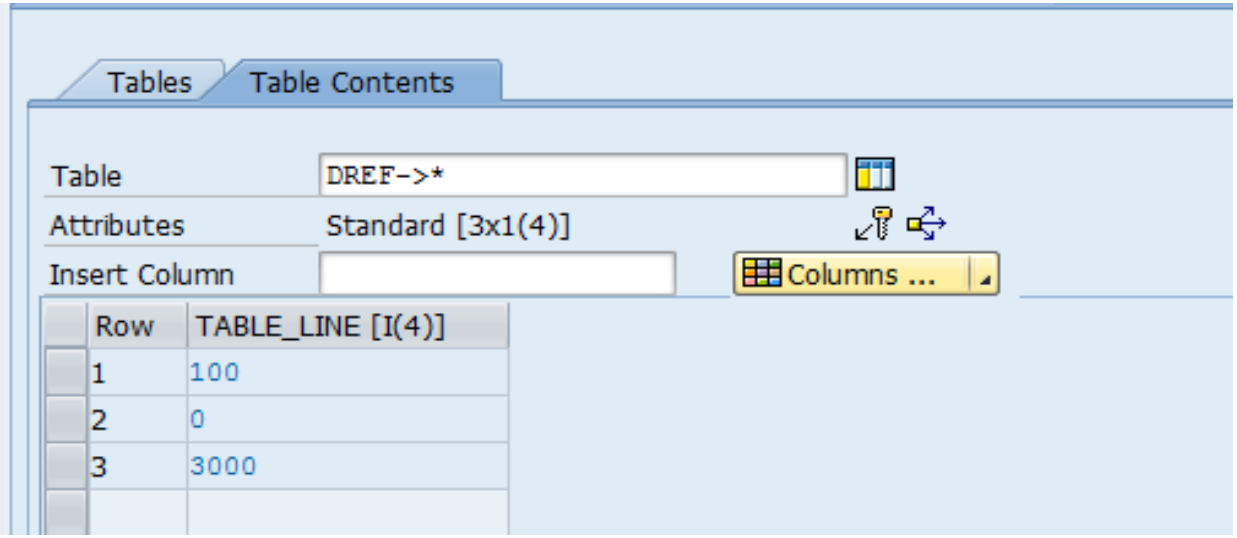
Output in debug mode:



Standard internal table declaration

TYPES t_itab TYPE STANDARD TABLE OF i WITH DEFAULT KEY.
DATA(dref) = NEW t_itab((100) ( ) (3000)).

Output in debug mode:

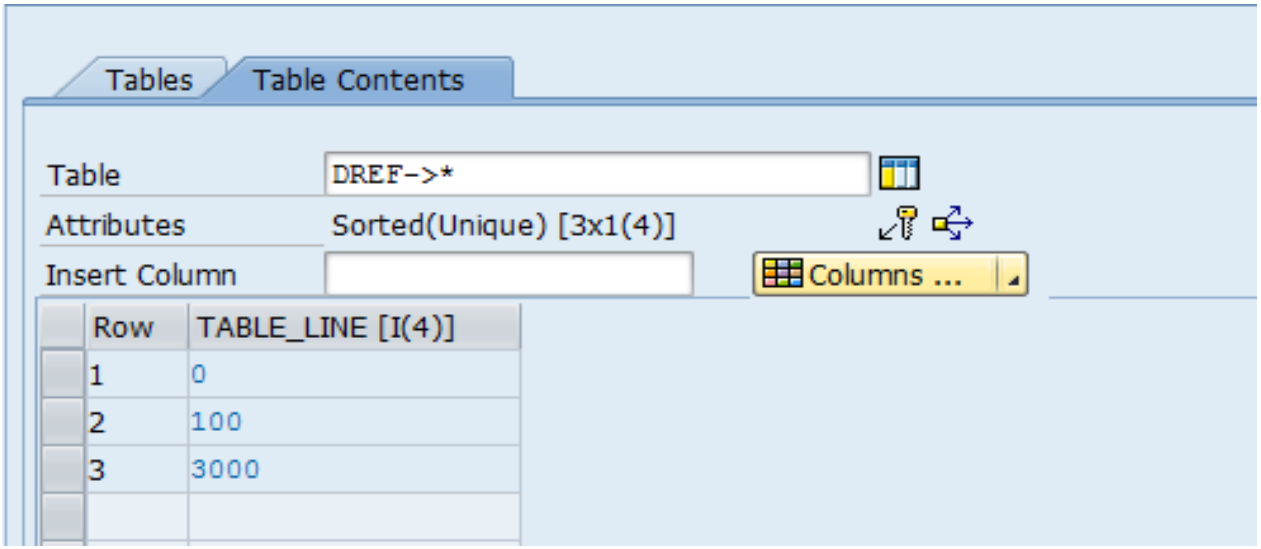


Here as we have declared the internal table as standard table so the values stored as 100->0->3000

Sorted internal table declaration

TYPES t_itab TYPE SORTED TABLE OF i WITH UNIQUE KEY **table_line**.
DATA(dref) = NEW t_itab((100) ( ) (3000)).

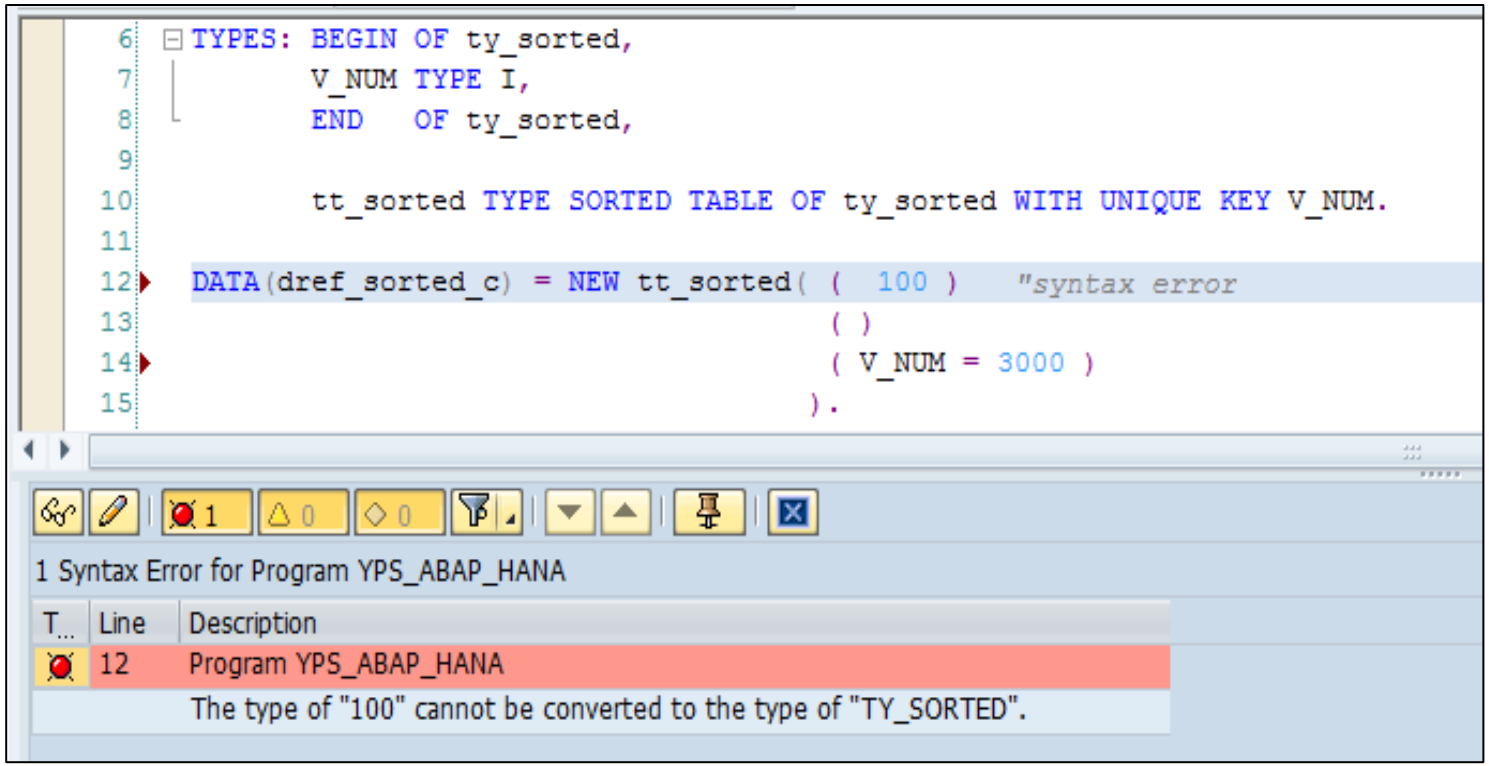
Output in debug mode:



Here as we have declared the internal table as sorted table so the values stored as 0->100->3000

Sorted internal table declaration

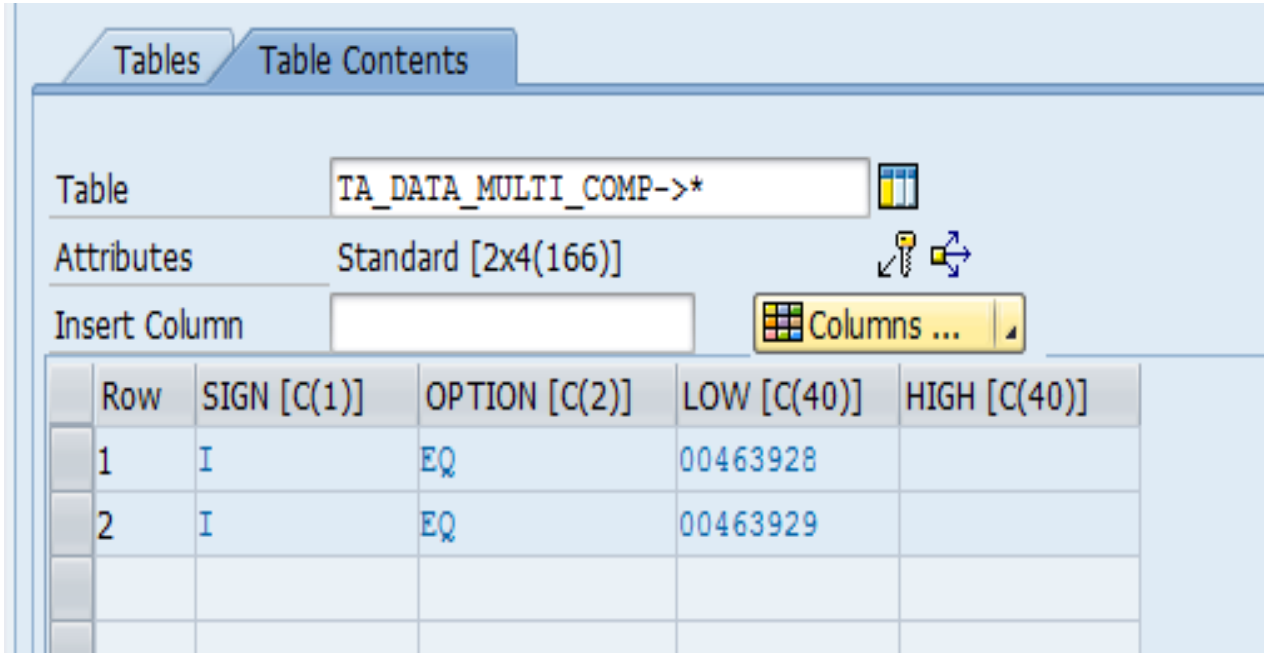
If you declared some specific component in type then you have to write ` Component = ` in new statement otherwise you will get an error.



Internal table with more components

```
TYPES: tt_data TYPE  MD_RANGE_T_MATNR.

DATA(ta_data_multi_comp) =
NEW tt_data( ( sign = 'I' Option = 'EQ' low = '00463928'  )
                    ( sign = 'I' Option = 'EQ' low = '00463929'  )  ).
```



How to work with deep structure

```
TYPES: BEGIN OF ty_alv_data,
         kunnr    TYPE kunnr,
         name1    TYPE name1,
         ort01    TYPE ort01,
         land1    TYPE land1,
         t_color TYPE lvc_t_scol, "structure
       END OF ty_alv_data.
TYPES: tt_alv_data TYPE STANDARD TABLE OF ty_alv_data WITH DEFAULT KEY.
   DATA(o_alv_data) = NEW tt_alv_data(
                                                    ( Build 1st row
                                                    ( Build inner rows i.e for
t_color ) )
                                                    ( Build 2nd row
                                                    ( Build inner rows i.e for
t_color ) )
                                                     )
```

How to work with deep structure

Code Snippet for Deep Structure

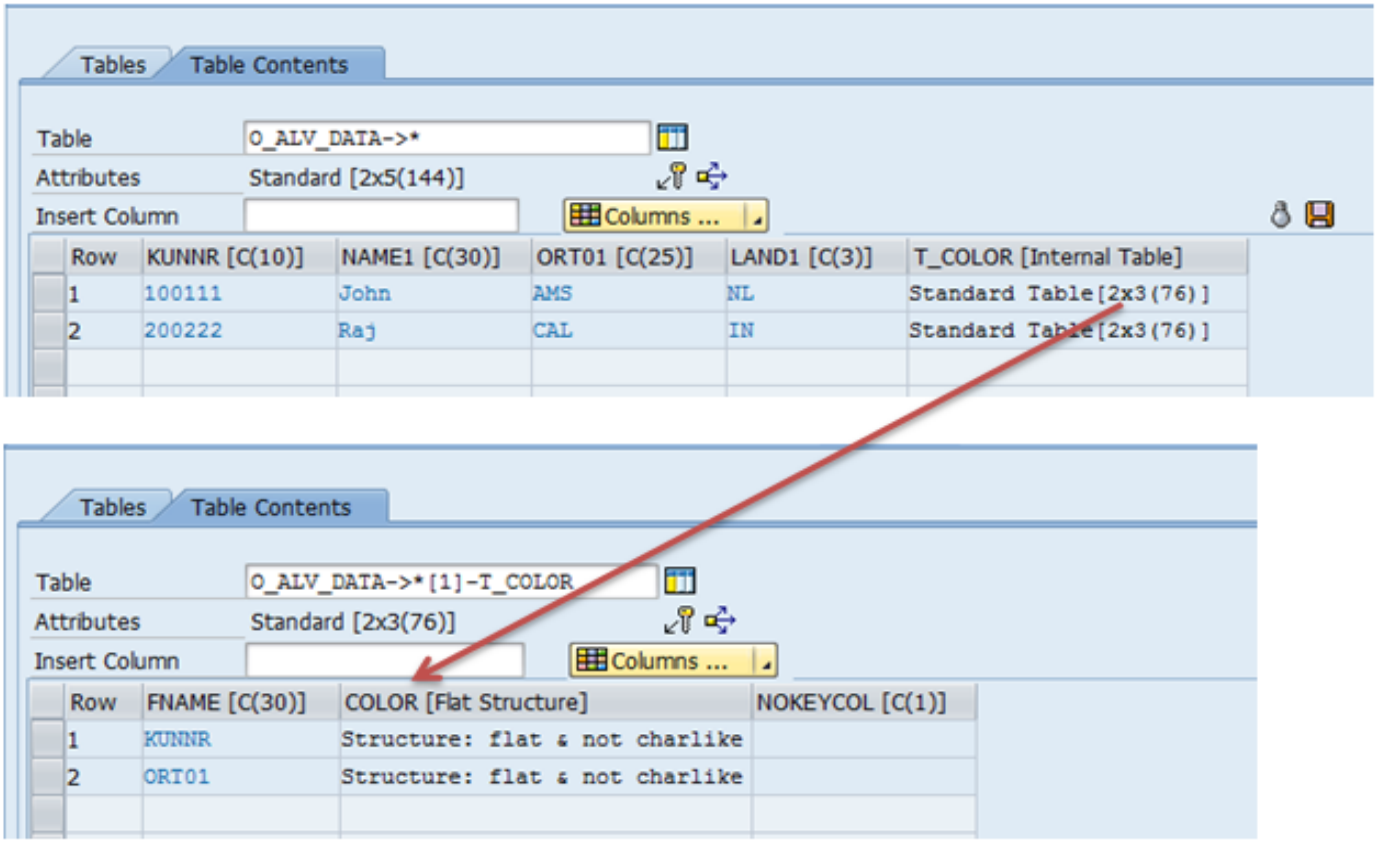
Field t_color is again a structure

```
DATA(o_alv_data) = NEW tt_alv_data(
            "First Row.....
            ( kunnr = '100111' name1 = 'John'
              ort01 = 'AMS' land1 = 'NL'
              " color table
              t_color = VALUE #(
                          " Color table - First Row
                          ( fname = 'KUNNR'
                            color-col = col_negative
                            color-int = 0
                            color-inv = 0
                          )
                          " Color Table - 2nd Row
                          ( fname = 'ORT01'
                            color-col = col_total
                            color-int = 1
                            color-inv = 1
                          )
                        )
            )
```

```
"Second row.....
            ( kunnr = '200222' name1 = 'Raj'
              ort01 = 'CAL' land1 = 'IN'
                          t_color = VALUE #(
                          " Color table - First Row
                          ( fname = 'KUNNR'
                            color-col = col_negative
                            color-int = 0
                            color-inv = 0
                          )
                          " Color Table - 2nd Row
                          ( fname = 'ORT01'
                            color-col = col_total
                            color-int = 1
                            color-inv = 1
                          )
                        )
            )
).
```

How to work with deep structure

Output in debug mode:



MOVE-CORRESPONDING for Internal Tables

You can use MOVE-CORRESPONDING not only for structures but also for internal tables now. Components of the same name are assigned row by row.

New additions EXPANDING NESTED TABLES and KEEPING TARGET LINES allow to resolve tabular components of structures and to append lines instead of overwriting existing lines.

Example:

```
MOVE-CORRESPONDING itab1 TO itab2 EXPANDING NESTED TABLES
                                  KEEPING TARGET LINES.
```

Table expressions

Table expressions replace READ TABLE statement

You need to use the square bracket [ ]. Within the bracket, you would need to specify the component you want to use as the key.

When table entry doesn’t exist, a catchable exception CX_SY_ITAB_LINE_NOT_FOUND is raised.

Demo program attached.

Table expressions

Demo Code Snippet

```
TYPES: tt_data TYPE_md_range_t_matnr. "standard tabl
etype

** Using New range table for matnr
DATA(ta_data_multi_comp) =   NEW tt_data( ).
data : tp_matnr type matnr.
SELECT * FROM mara UP TO 5 ROWS
        INTO TABLE @DATA(mara) " Host variable with esc
ape character @
        WHERE matnr IN @ta_data_multi_comp->* .

SELECT matnr, maktx FROM makt
   INTO TABLE @DATA(ta_makt)
  FOR ALL ENTRIES IN @mara
  WHERE matnr = @mara-matnr.

loop at mara into data(wa).
try.
 data(tp_matnr1) =  ta_makt[ matnr = wa-matnr ]-
matnr. " Substitute of READ
  write: / tp_matnr1 .
  CATCH cx_sy_itab_line_not_found.
    endtry.
endloop.
```

GROUP BY clause for Internal Tables

GROUP BY replace AT NEW or other means of going through grouped data

```
LOOP AT flights INTO DATA(flight)
    GROUP BY ( carrier = flight-carrid cityfr = flight-cityfrom )
             ASCENDING
             ASSIGNING FIELD-SYMBOL(<group>).
  CLEAR members.
  LOOP AT GROUP <group> ASSIGNING FIELD-SYMBOL(<flight>).
    members = VALUE #( BASE members ( <flight> ) ).
  ENDLOOP.
```

Looks like dreaded nested LOOPs, but it isn't quite that – no quadratic behavior! What happens here is that the first LOOP statement is executed over all internal table lines in one go and the new GROUP BY addition groups the lines. Technically, the lines are bound internally to a group that belongs to a group key that is specified behind GROUP BY.

Demo program attached.

FILTER expressions

The new FILTER operator enables two kinds of filtering an internal table

i. Filter with single values

ii. Filter with filter table

Filter with single values: Simply extract the lines from an internal table into a tabular result, that fulfill a simple value condition.

```
DATA(extract) =  FILTER #( spfli_tab USING KEY carr_city
                           WHERE carrid    = CONV #( to_upper( carrid ) ) AND
                           cityfrom = CONV #( to_upper( cityfrom ) ) ).
```

Note: As a prerequisite, the filtered table (spfli_tab) **must** have a sorted or a hash key (primary or secondary), that is evaluated behind WHERE

FILTER expressions

Filter with filter table: Compare the lines of one table with the contents of another table, the filter table, and you extract those lines, where at least one match is found

```
TYPES: BEGIN OF filter,
         cityfrom TYPE spfli-cityfrom,
         cityto   TYPE spfli-cityto,
       END OF filter,
       filter_tab TYPE HASHED TABLE OF filter
                      WITH UNIQUE KEY cityfrom cityto.

  DATA(filter_tab) = …

  DATA(extract) = FILTER #( spfli_tab IN filter_tab
                    WHERE cityfrom = cityfrom  AND cityto = cityto ).
```

Note: Here, the filter table – that can be specified also as a functional method call – must have a sorted or a hashed key (primary or secondary) that is evaluated.

INNER JOIN Improvement

You can use wildcard like SELECT * in new inner join

Old syntax

```
SELECT a~vbeln b~posnr b~matnr FROM vbak AS a INNER JOIN b AS vbap
                                        ON a~vbeln = b~vbeln
                                        INTO TABLE li_vbeln
                                        WHERE a~auart = 'Z1IN'.
```

New syntax:

```
SELECT a~*, b~posnr, b~matnr FROM vbak AS a INNER JOIN vbap as b
                                        ON a~vbeln = b~vbeln
                                        WHERE a~auart = 'Z1IN'
                                        INTO TABLE @DATA(li_vbeln).
```

Note: The symbol * (asterisk) it acts just like the wildcard SELECT * , and for this sample you will get all fields in VBAK table.

NEW keyword for creating Objects

You can use ‘NEW’ to instances an object instead of use CREATE OBJECT.

Old syntax

```
DATA : lo_myclass TYPE REF TO ZCL_MYCLASS.
CREATE OBJECT lo_myclass EXPORTING myname = 'India'.
```

New syntax:

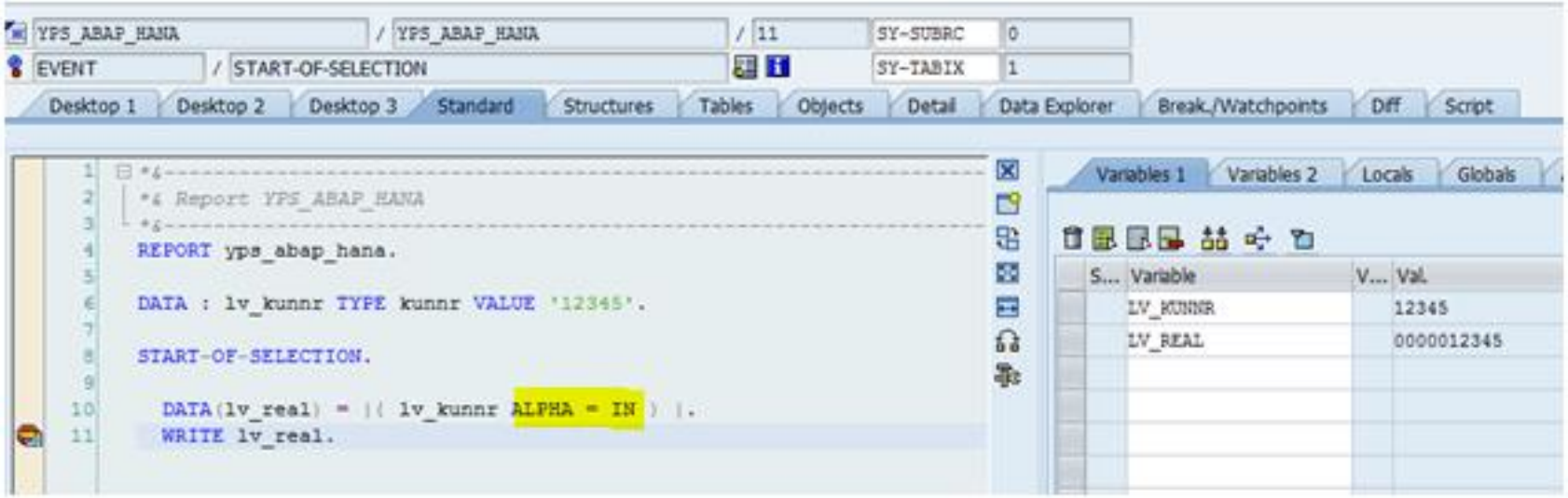
```
lo_myclass = NEW zcl_Myclass( myname = 'India' ).
```

Note: Key word ‘NEW’ is used to create instance of class ZCL_MYCLASS, Here lo_myclass is the object name.

CONVERSION_EXIT_ALPHA_INPUT/OURPUT

You don’t need use CONVERSION_EXIT_ALPHA_INPUT and CONVERSION_EXIT_ALPHA_OUTPUT;

You just need ALPHA keyword formatting option with OUT or IN.



KUNNR value of ‘12345’ changes to ‘000001235’, 5 zero added as KUNNR length is 10 CHAR

Using SWITCH statement

Using SWITCH statement as replacement of CASE

Using CASE you need to keep mentioning what variable you're filling in every branch in CASE Statement but If you use SWITCH statement, you don't need to do it

Case statement:

```
CASE LV_INDICATOR.
        WHEN 1.  LV_DAY = 'January'.
        WHEN 2.   LV_DAY = 'February'.
  ENDCASE.
```

Switch statement:

```
DATA(lv_day) = SWITCH char10( lv_indicator
                              WHEN 1 THEN 'January`
                              WHEN 2 THEN 'February' ).
```

Note: Using *SWITCH statement, you don't need mention LV_DAY variable in every branch*

Summary

- We have learned ABAP New syntax (SAP NW 7.4 onwards)
- Some new key word like FILTER expression, NEW, Table expression.

Web Link:

https://blogs.sap.com/2016/03/02/old-and-new-abap-syntax-overview-sheet/

http://www.saptutorial.org/new-abap-language-in-abap-7-4/