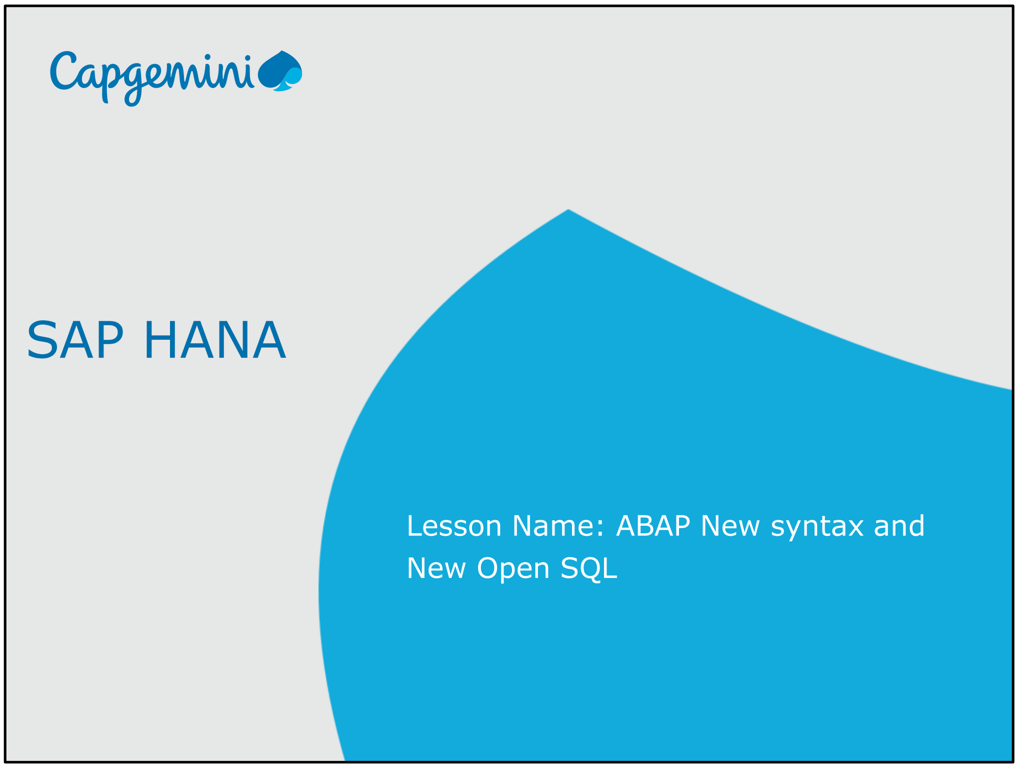
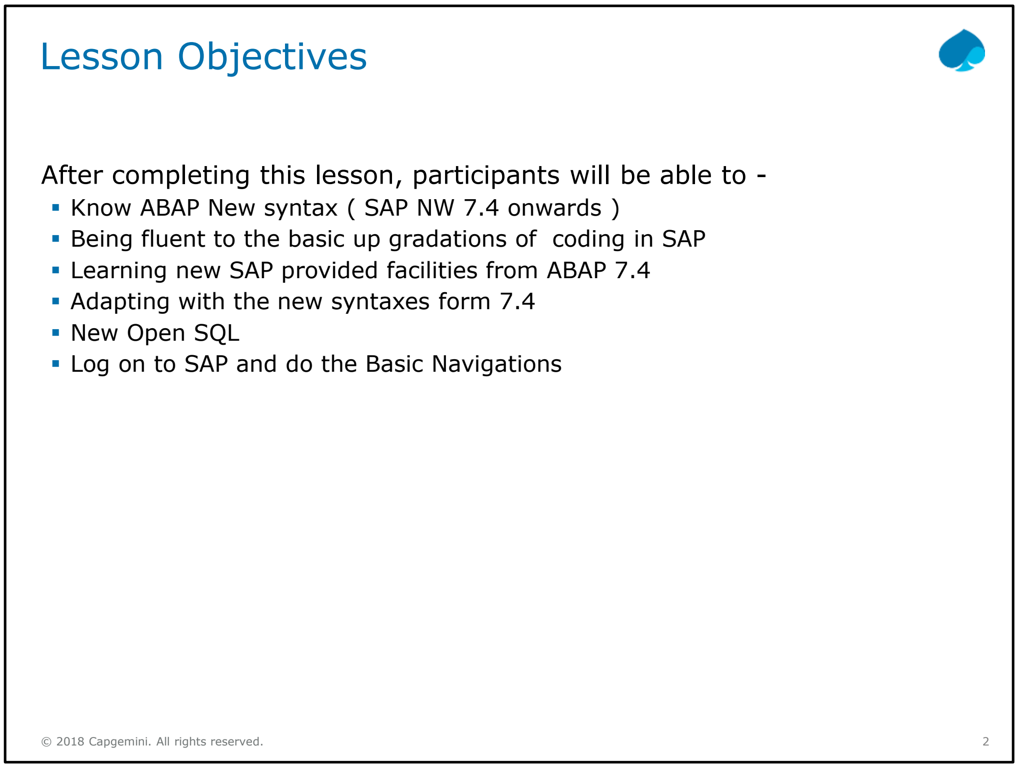


SAP HANA

Lesson Name: ABAP New syntax and New Open SQL

ABAP New syntax (SAP NW 7.4 onwards)

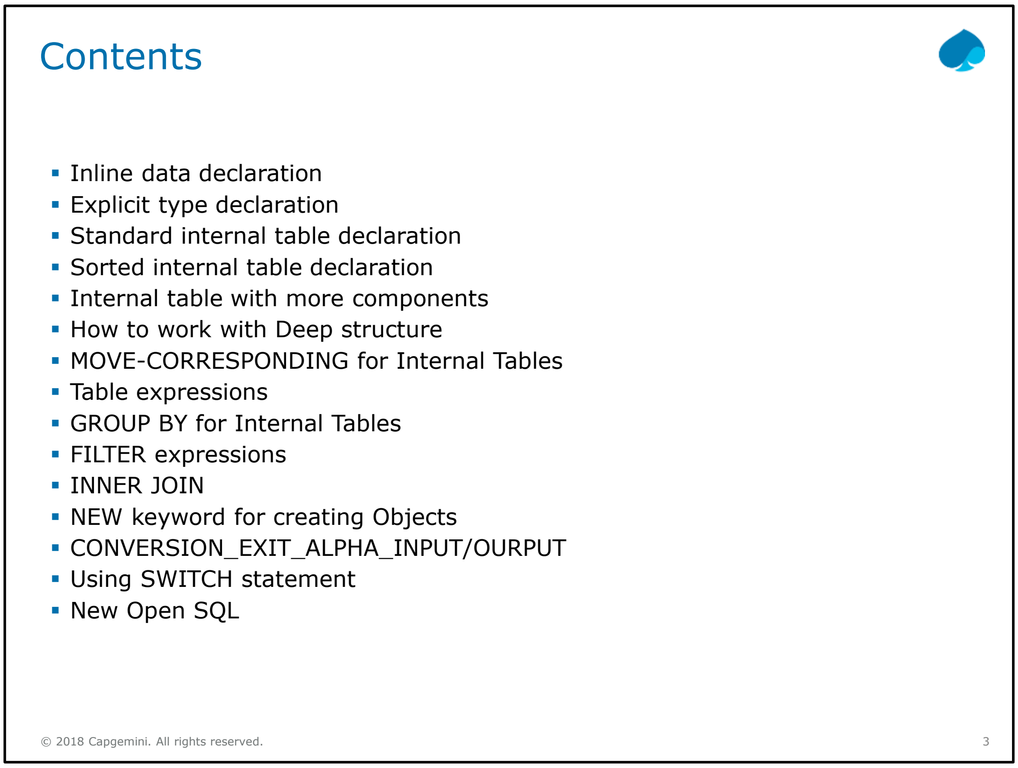


Lesson Objectives

After completing this lesson, participants will be able to -

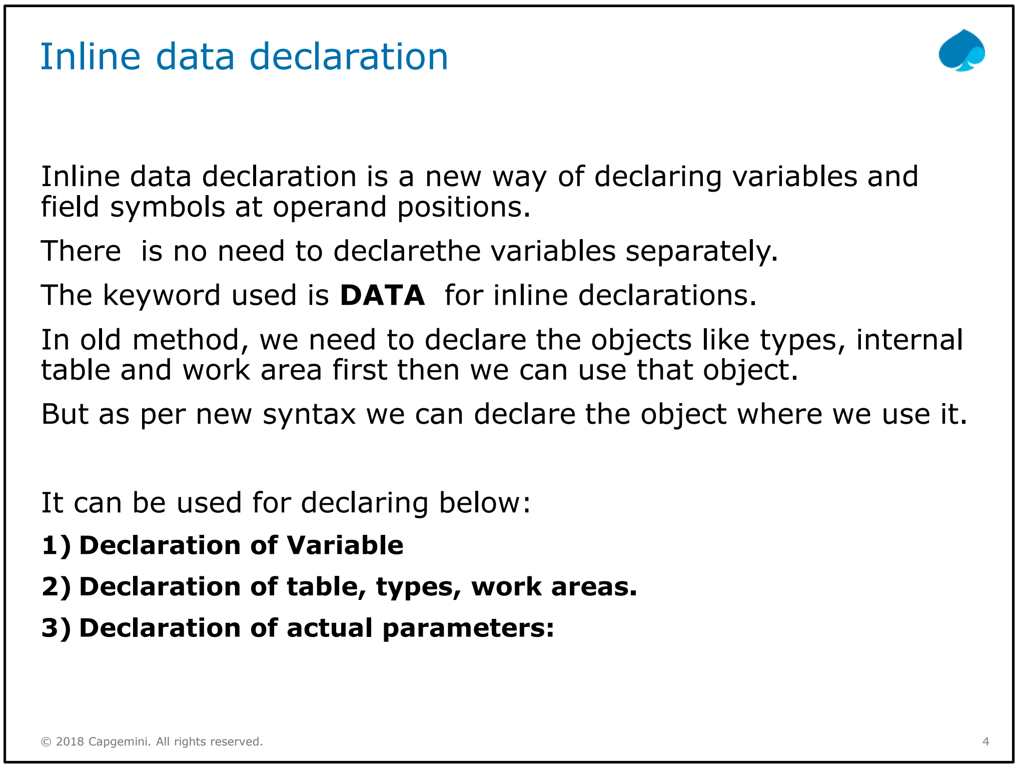
- Know ABAP New syntax (SAP NW 7.4 onwards)
- Being fluent to the basic up gradations of coding in SAP
- Learning new SAP provided facilities from ABAP 7.4
- Adapting with the new syntaxes form 7.4
- New Open SQL
- Log on to SAP and do the Basic Navigations

2



Contents

3



Inline data declaration

Inline data declaration is a new way of declaring variables and field symbols at operand positions.

There is no need to declarethe variables separately.

The keyword used is **DATA** for inline declarations.

In old method, we need to declare the objects like types, internal table and work area first then we can use that object.

But as per new syntax we can declare the object where we use it.

It can be used for declaring below:

1) Declaration of Variable

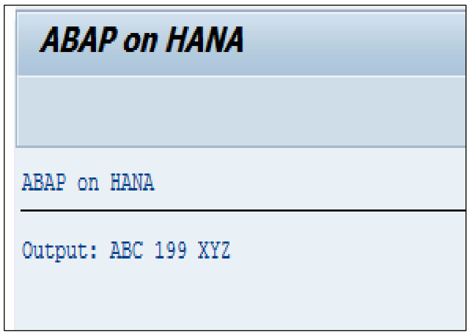
2) Declaration of table, types, work areas.

3) Declaration of actual parameters:

Inline data declaration

1) Declaration of Variable

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

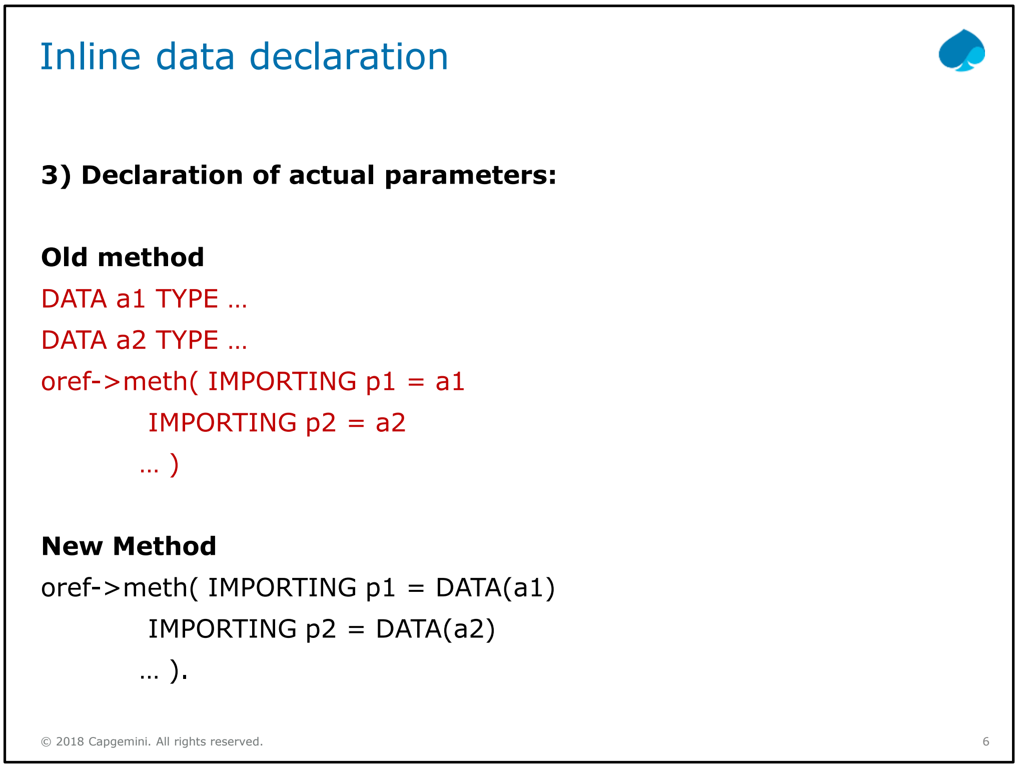


```
ABAP on HANA

Output: ABC 199 XYZ
```

2) Declaration of work areas:

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

5



Inline data declaration

3) 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)
           … ).
```

6

In old method we need to declare the object like types, Internal table and work area first then we can use that object.
But as per new syntax we can declare the object where we use it.

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:

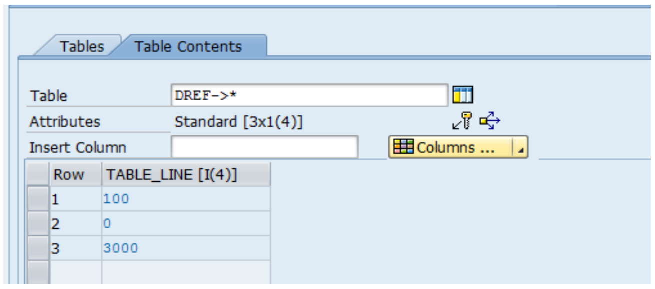


Table DREF->*
Attributes Standard [3x1(4)]
Insert Column

Row	TABLE_LINE [I(4)]
1	100
2	0
3	3000

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

7

Sorted internal table declaration

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

The keyword **TABLE_LINE** is used for dynamic table /Variable insert

(

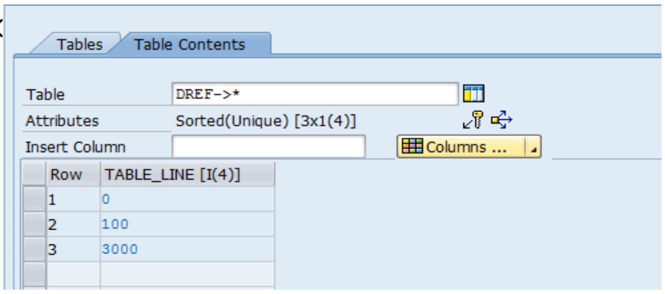


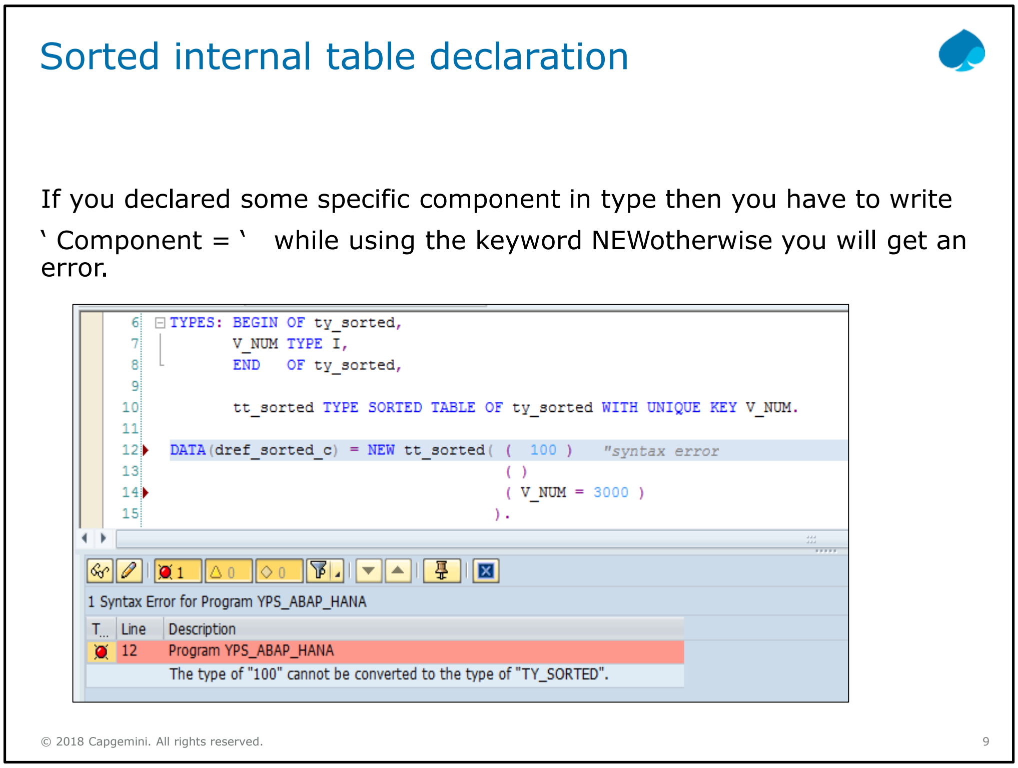
Table	DREF->*
Attributes	Sorted(Unique) [3x1(4)]
Insert Column	

Row	TABLE_LINE [I(4)]
1	0
2	100
3	3000

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

8

** TABLE_LINE ---> Line for dynamic table /Variable insert



Sorted internal table declaration

If you declared some specific component in type then you have to write ‵ Component = ‵ while using the keyword NEWotherwise you will get an error.

```
 6  TYPES: BEGIN OF ty_sorted,
 7          V_NUM TYPE I,
 8          END   OF ty_sorted,
 9
10          tt_sorted TYPE SORTED TABLE OF ty_sorted WITH UNIQUE KEY V_NUM.
11
12  DATA(dref_sorted_c) = NEW tt_sorted( (  100 )    "syntax error
13                                       ( )
14                                       ( V_NUM = 3000 )
15                                      ).
```

1 Syntax Error for Program YPS_ABAP_HANA

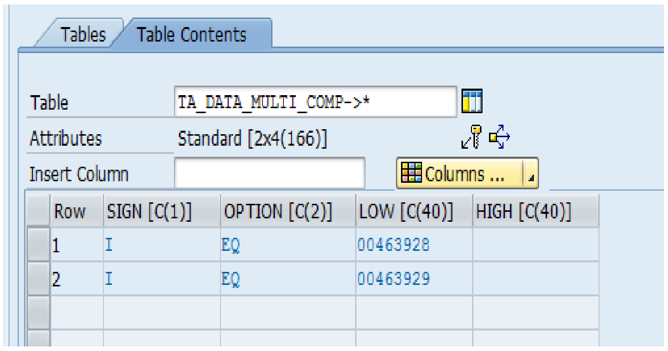
T...	Line	Description
●	12	Program YPS_ABAP_HANA
		The type of "100" cannot be converted to the type of "TY_SORTED".

9

```
** TABLE_LINE ---> Line for dynamic table /Variable insert
** No syntax error when V_NUM component assign
TYPES: BEGIN OF ty_sorted,
       V_NUM TYPE I,
       END   OF ty_sorted,

       tt_sorted TYPE SORTED TABLE OF ty_sorted WITH UNIQUE KEY V_NUM.

DATA(dref_sorted_c) = NEW tt_sorted( ( V_NUM = 100 )
                                     ( )
                                     ( V_NUM = 3000 )
                                    ).
```

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'  )  ).
```



Tables | Table Contents

Table	TA_DATA_MULTI_COMP->*
Attributes	Standard [2x4(166)]
Insert Column	

Row	SIGN [C(1)]	OPTION [C(2)]	LOW [C(40)]	HIGH [C(40)]
1	I	EQ	00463928	
2	I	EQ	00463929	

10

MD_RANGE_T_MATNR is Standard tabletype
ta_data_multi_com is multi component internal table

MOVE-CORRESPONDING for Internal Tables

You can use MOVE-CORRESPONDING not only for structures but also for internal tables . 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:

OLD :

```
MOVE-CORRESPONDING wa1 TO wa2.
```

New:

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

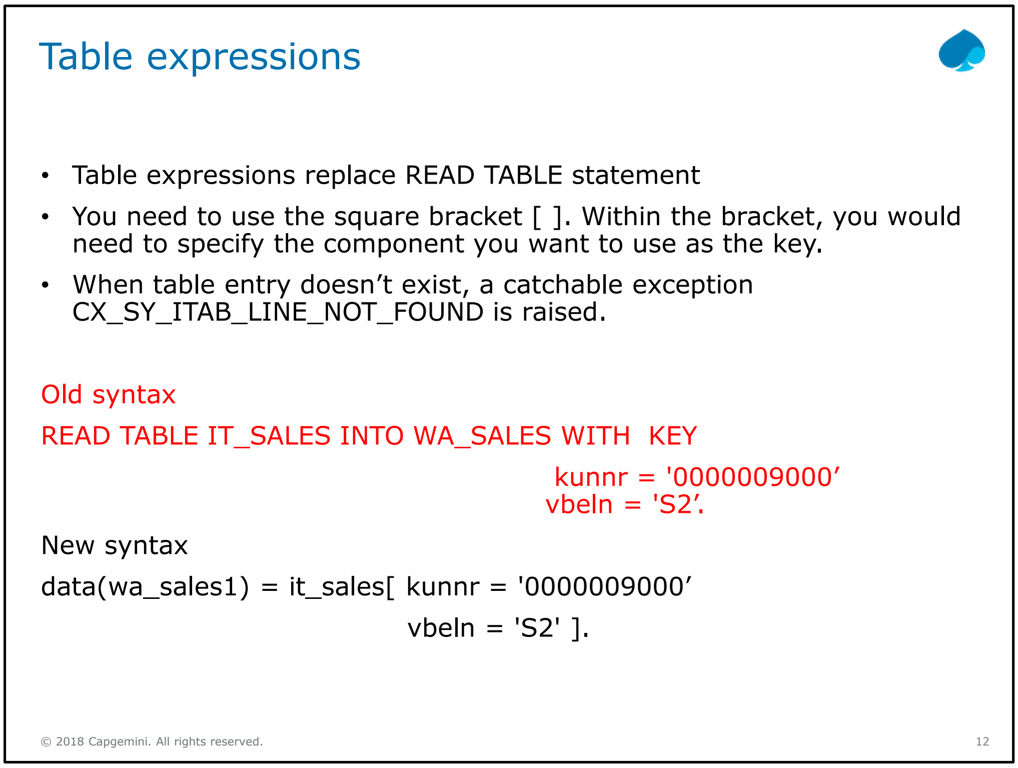


Table expressions

- 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.

Old syntax

```
READ TABLE IT_SALES INTO WA_SALES WITH  KEY
                                              kunnr = '0000009000’
                                             vbeln = 'S2’.
```

New syntax

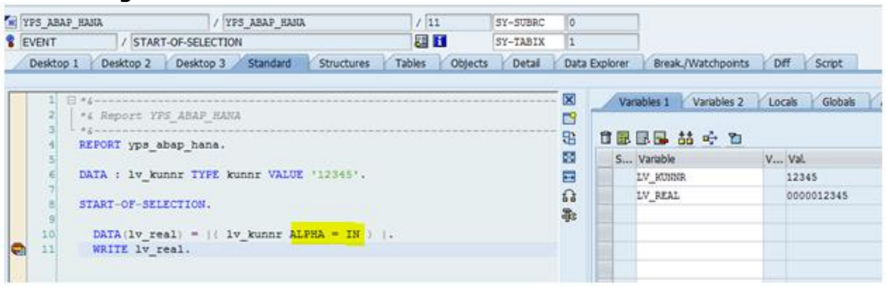
```
data(wa_sales1) = it_sales[ kunnr = '0000009000’
                            vbeln = 'S2' ].
```

CONVERSION_EXIT_ALPHA_INPUT/OURPUT

OLD : Traditionally the function modules CONVERSION_EXIT_ALPHA_INPUT and CONVERSION_EXIT_ALPHA_OUTPUT were used for conversion

New : You just need to use the **ALPHA** keyword formatting option with OUT or IN.

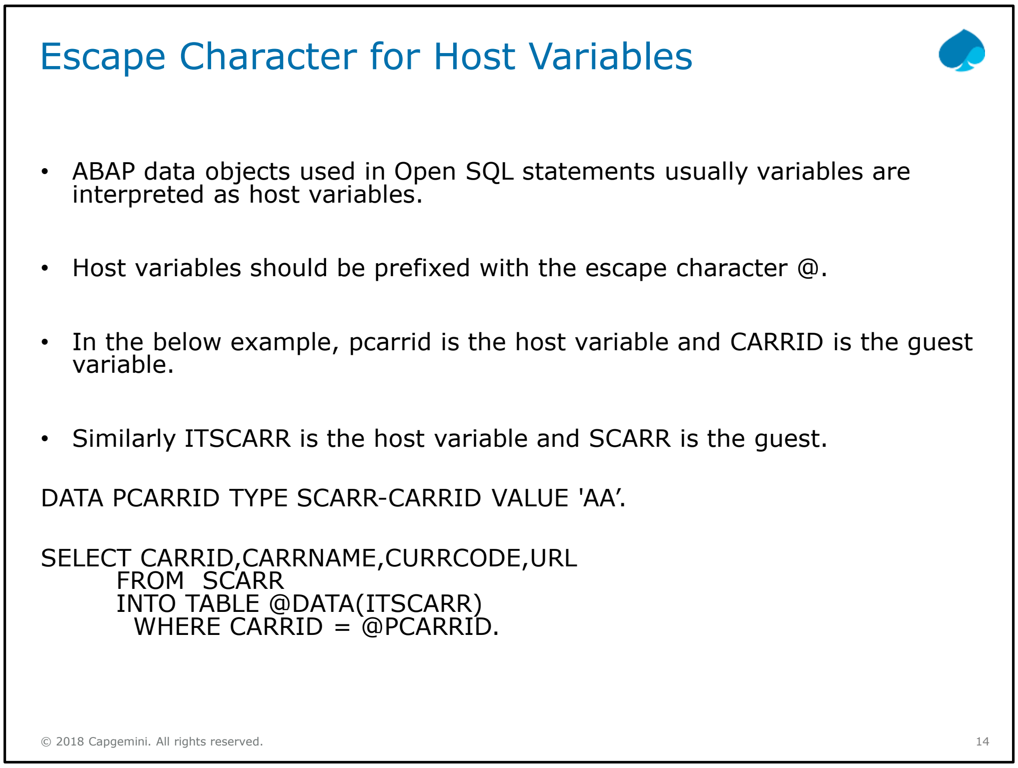
Eg : KUNNR value of '12345' changes to '000001235', 5 zero added as KUNNR length is 10 CHAR



```
YPS_ABAP_HANA / YPS_ABAP_HANA / 11      SY-SUBRC 0
EVENT / START-OF-SELECTION              SY-TABIX 1
```

```
*&---------------------------------------------------------------------
*& Report YPS_ABAP_HANA
*&---------------------------------------------------------------------
REPORT yps_abap_hana.

DATA : lv_kunnr TYPE kunnr VALUE '12345'.

START-OF-SELECTION.

  DATA(lv_real) = |{ lv_kunnr ALPHA = IN }|.
  WRITE lv_real.
```

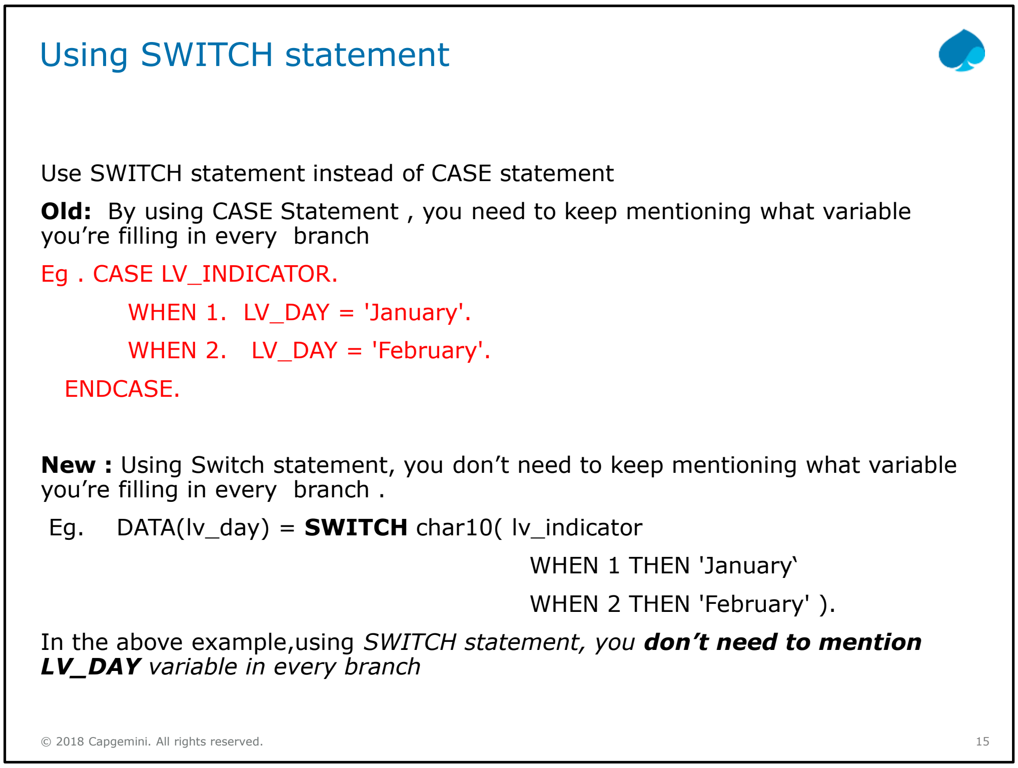
S...	Variable	V...	Val.
	LV_KUNNR		12345
	LV_REAL		0000012345



Escape Character for Host Variables

- ABAP data objects used in Open SQL statements usually variables are interpreted as host variables.
- Host variables should be prefixed with the escape character @.
- In the below example, pcarrid is the host variable and CARRID is the guest variable.
- Similarly ITSCARR is the host variable and SCARR is the guest.

```
DATA PCARRID TYPE SCARR-CARRID VALUE 'AA'.

SELECT CARRID,CARRNAME,CURRCODE,URL
        FROM  SCARR
        INTO TABLE @DATA(ITSCARR)
          WHERE CARRID = @PCARRID.
```



Using SWITCH statement

Use SWITCH statement instead of CASE statement

Old: By using CASE Statement , you need to keep mentioning what variable you’re filling in every branch

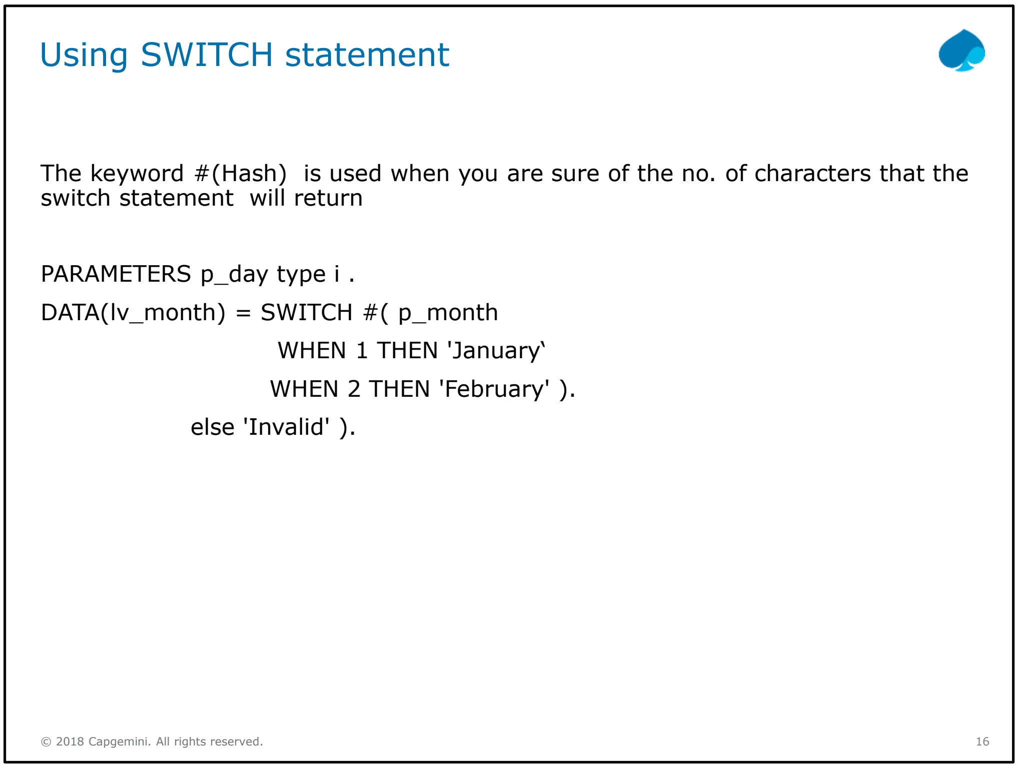
```
Eg . CASE LV_INDICATOR.
          WHEN 1.  LV_DAY = 'January'.
          WHEN 2.   LV_DAY = 'February'.
  ENDCASE.
```

New : Using Switch statement, you don’t need to keep mentioning what variable you’re filling in every branch .

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

In the above example,using *SWITCH statement, you* ***don’t need to mention LV_DAY*** *variable in every branch*

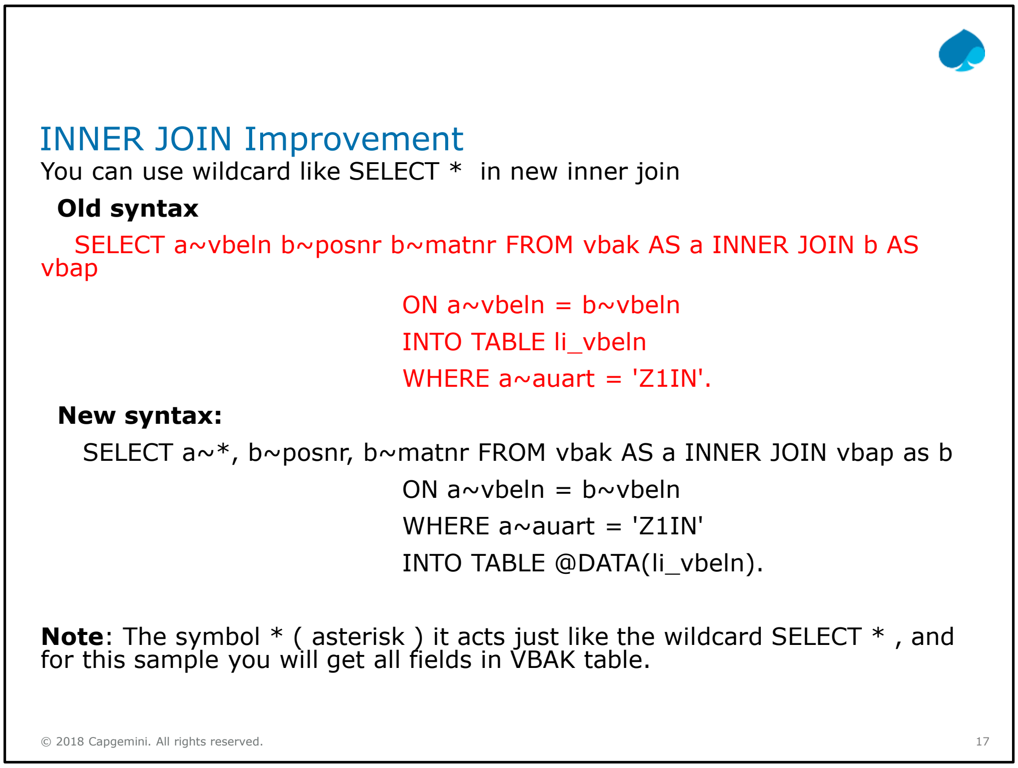
15



Using SWITCH statement

The keyword #(Hash) is used when you are sure of the no. of characters that the switch statement will return

```
PARAMETERS p_day type i .
DATA(lv_month) = SWITCH #( p_month
                              WHEN 1 THEN 'January`
                             WHEN 2 THEN 'February' ).
                   else 'Invalid' ).
```



INNER JOIN Improvement

You can use wildcard like SELECT * in new inner join

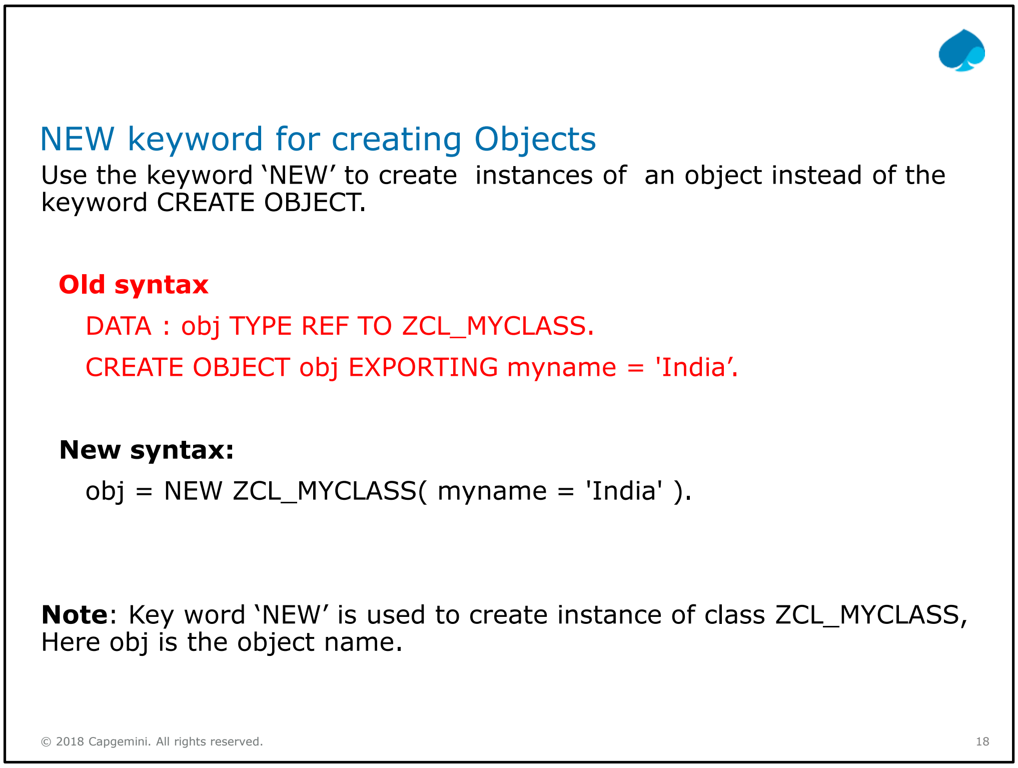
Old syntax

```
   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

Use the keyword ‘NEW’ to create instances of an object instead of the keyword CREATE OBJECT.

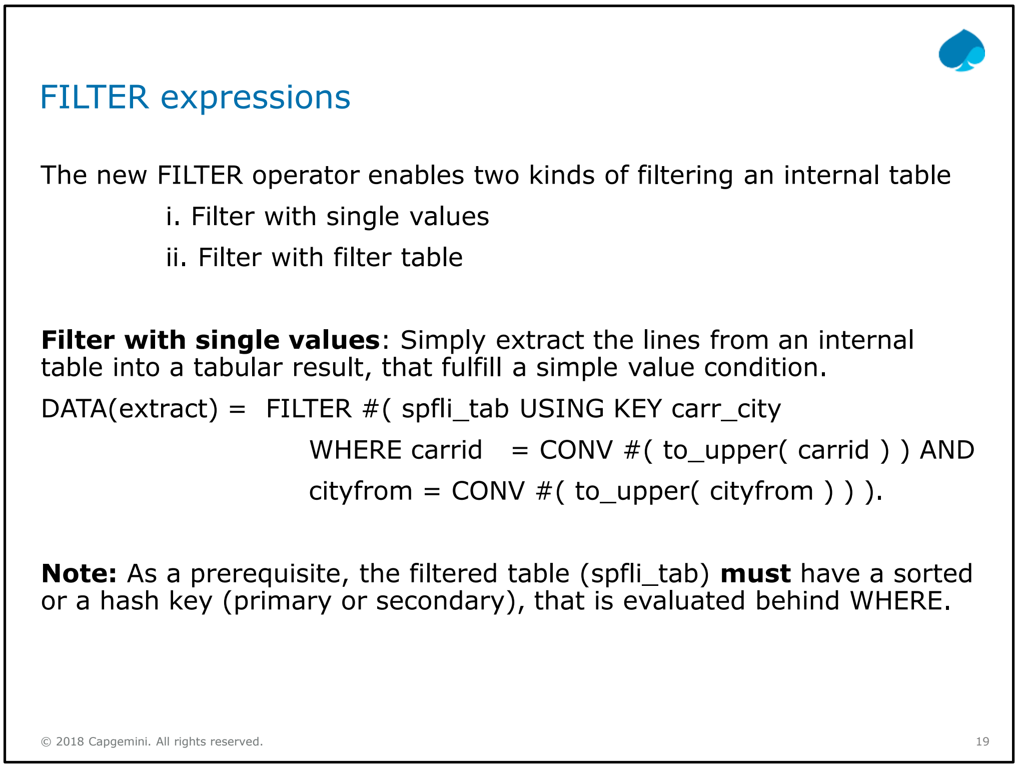
Old syntax

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

New syntax:

```
obj = NEW ZCL_MYCLASS( myname = 'India' ).
```

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



FILTER expressions

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

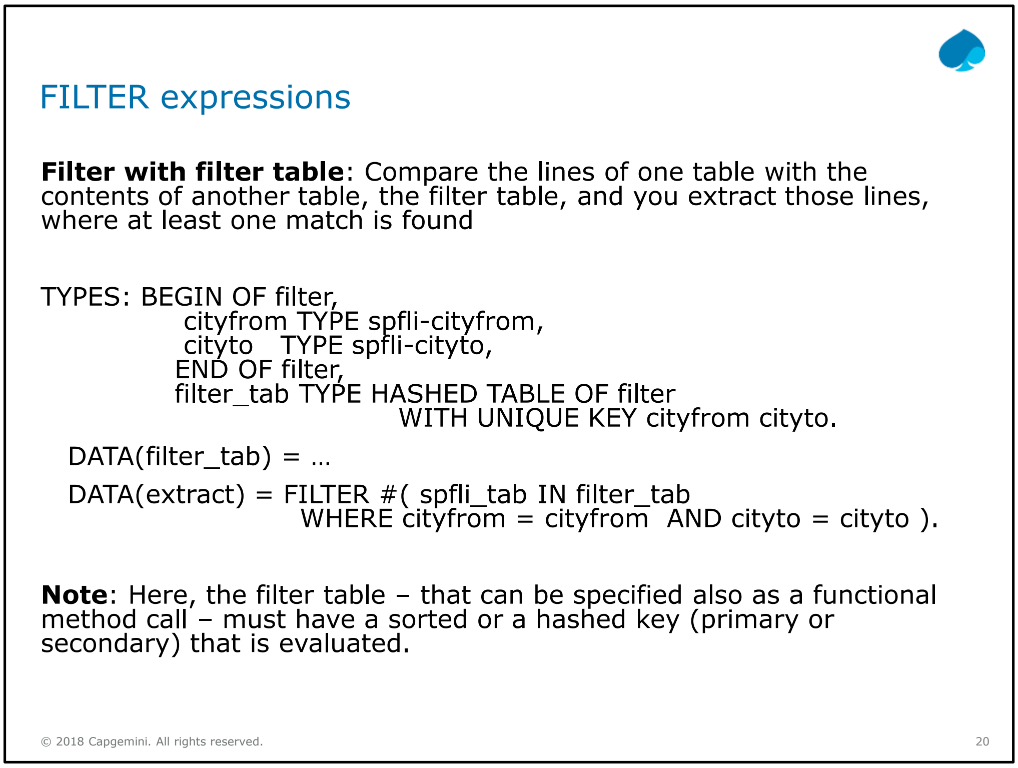
i. Filter with single values

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.

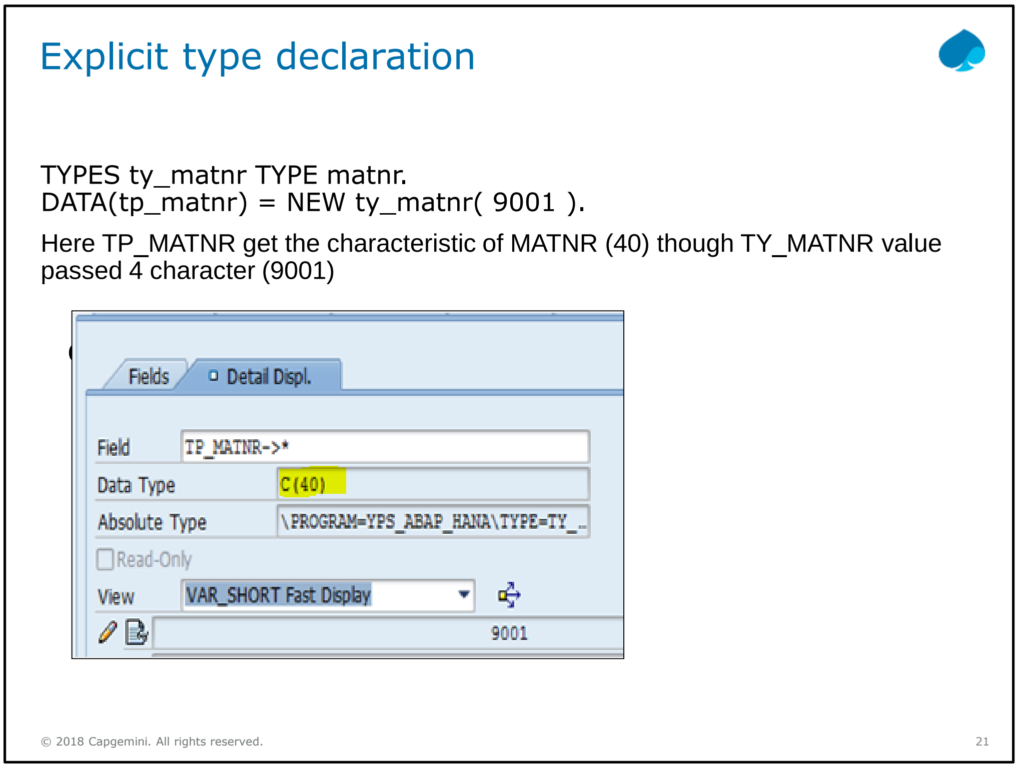


FILTER expressions

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.



Explicit type declaration

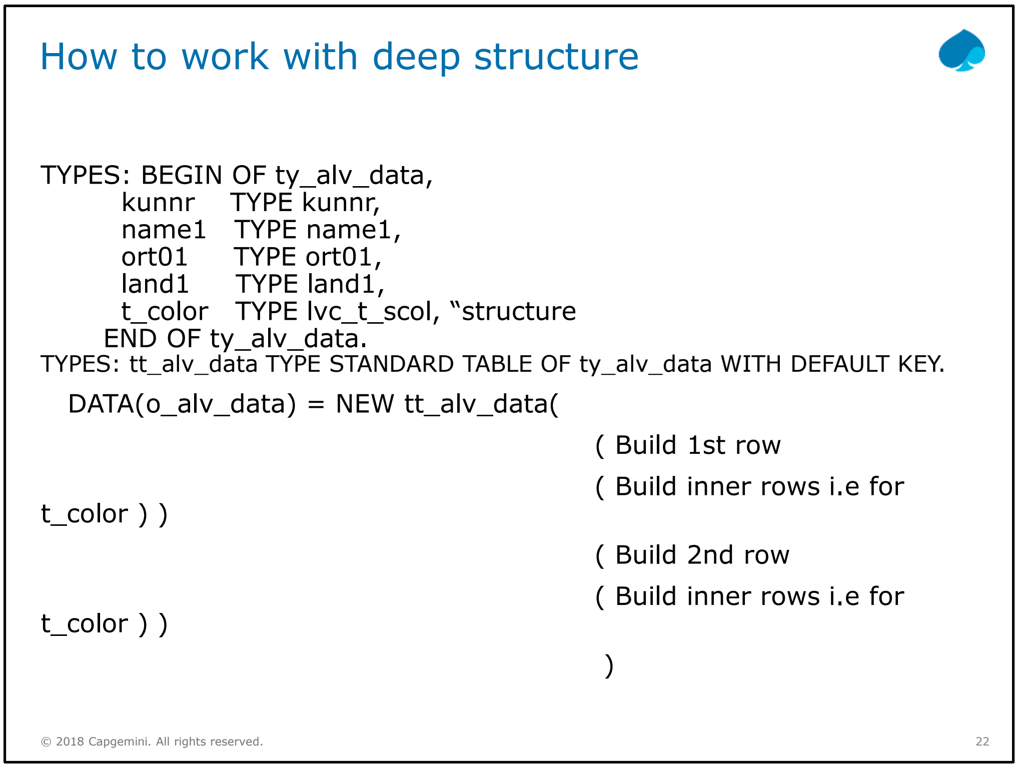
TYPES ty_matnr TYPE matnr.
DATA(tp_matnr) = NEW ty_matnr(9001).

Here TP_MATNR get the characteristic of MATNR (40) though TY_MATNR value passed 4 character (9001)

Field TP_MATNR->*
Data Type C(40)
Absolute Type \PROGRAM=YPS_ABAP_HANA\TYPE=TY_
Read-Only
View VAR_SHORT Fast Display
9001

21

Here TP_MATNR get the characteristic of MATNR (40) though TY_MATNR value passed 4 character (9001)



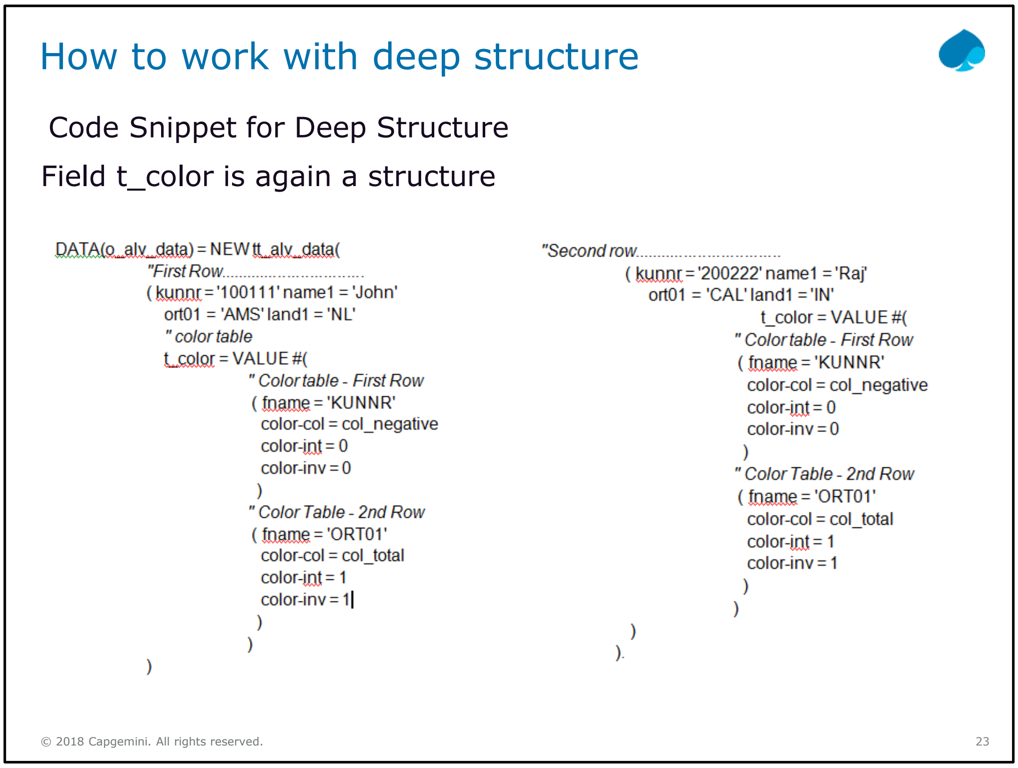
How to work with deep structure

```
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 ) )
                                                   )
```

22

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

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:

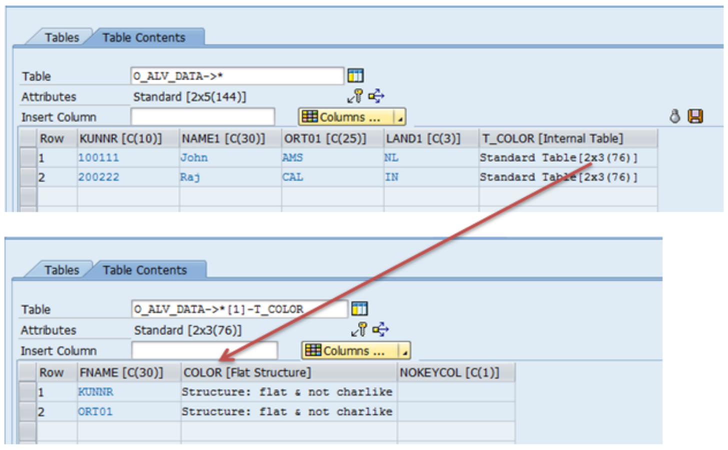


Table: O_ALV_DATA->*
Attributes: Standard [2x5(144)]
Insert Column:

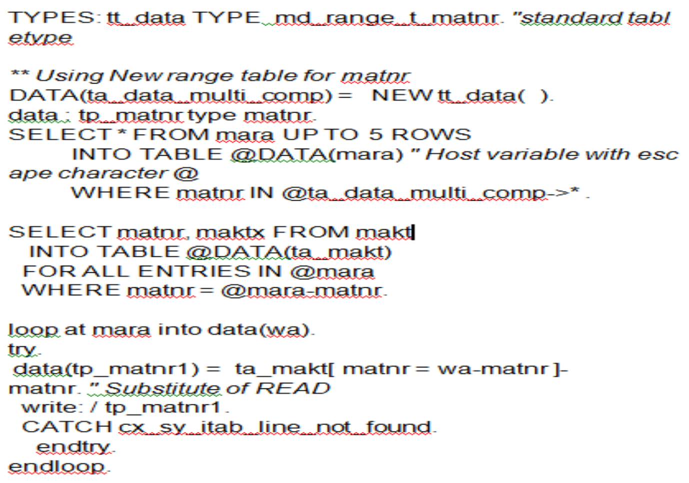
Row	KUNNR [C(10)]	NAME1 [C(30)]	ORT01 [C(25)]	LAND1 [C(3)]	T_COLOR [Internal Table]
1	100111	John	AMS	NL	Standard Table[2x3(76)]
2	200222	Raj	CAL	IN	Standard Table[2x3(76)]

Table: O_ALV_DATA->*[1]-T_COLOR
Attributes: Standard [2x3(76)]
Insert Column:

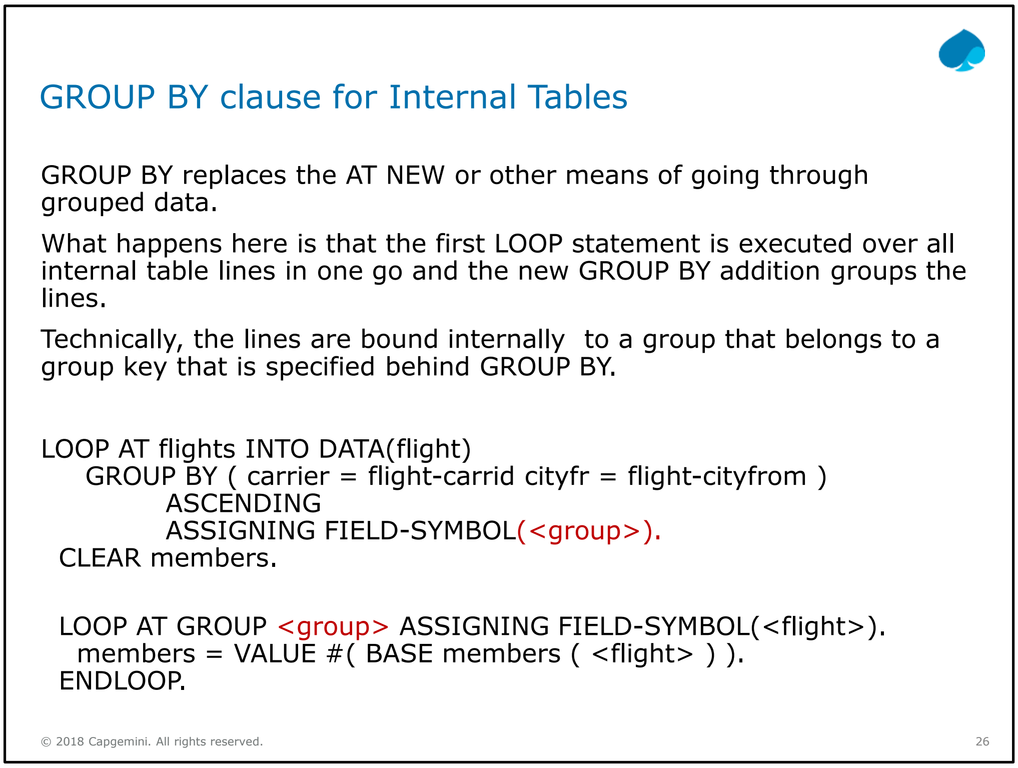
Row	FNAME [C(30)]	COLOR [Flat Structure]	NOKEYCOL [C(1)]
1	KUNNR	Structure: flat & not charlike	
2	ORT01	Structure: flat & not charlike	

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 replaces the AT NEW or other means of going through grouped data.

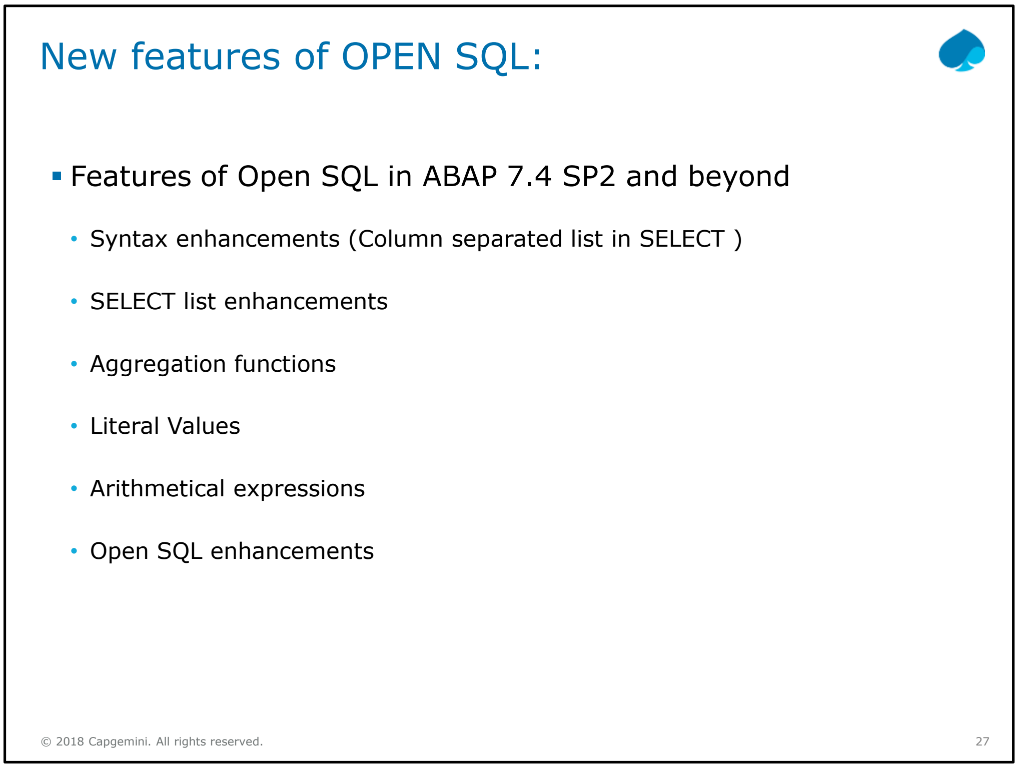
What happens here is that the first LOOP statement is executed over all internal table lines in one go and the new GROUP BY addition groups the lines.

Technically, the lines are bound internally to a group that belongs to a group key that is specified behind GROUP BY.

```
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.
```

26

```
DATA flights TYPE TABLE OF spfli WITH EMPTY KEY.

SELECT * FROM  spfli
        WHERE carrid = ''
        INTO TABLE @flights.

DATA members LIKE flights.
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.
  cl_demo_output=>write( members ).
ENDLOOP.
cl_demo_output=>display( ).
```



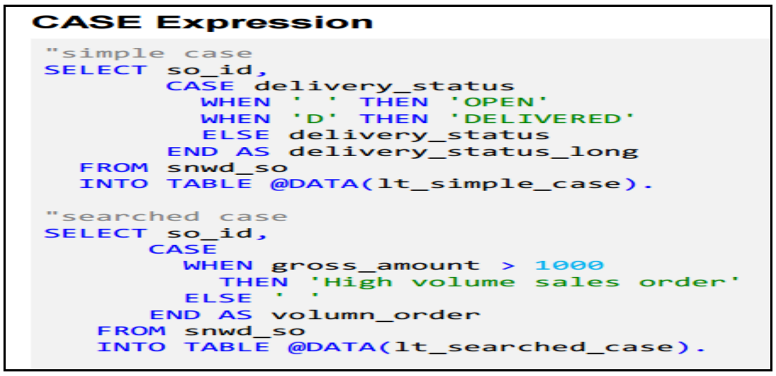
New features of OPEN SQL:

- Features of Open SQL in ABAP 7.4 SP2 and beyond
  - Syntax enhancements (Column separated list in SELECT)
  - SELECT list enhancements
  - Aggregation functions
  - Literal Values
  - Arithmetical expressions
  - Open SQL enhancements

New features of OPEN SQL:

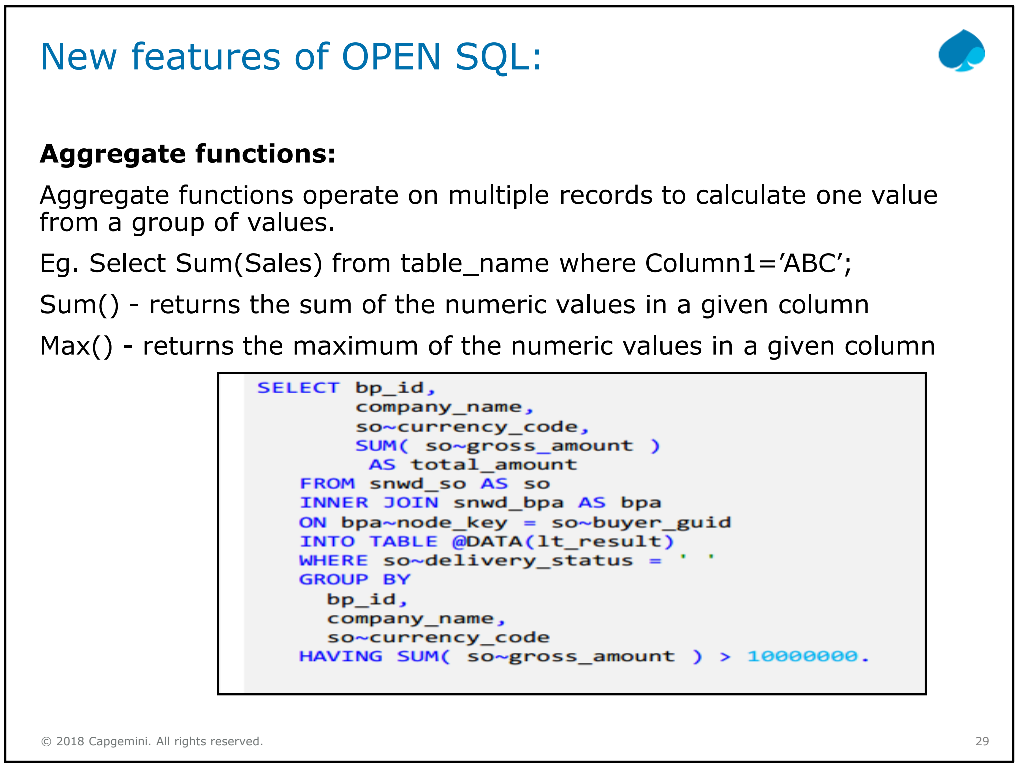
Select List enhancements:

- Conditional expressions like CASE statement can be used in Select .



CASE Expression

```
"simple case
SELECT so_id,
       CASE delivery_status
         WHEN ' ' THEN 'OPEN'
         WHEN 'D' THEN 'DELIVERED'
         ELSE delivery_status
       END AS delivery_status_long
  FROM snwd_so
  INTO TABLE @DATA(lt_simple_case).

"searched case
SELECT so_id,
      CASE
        WHEN gross_amount > 1000
          THEN 'High volume sales order'
        ELSE ' '
      END AS volumn_order
   FROM snwd_so
   INTO TABLE @DATA(lt_searched_case).
```



New features of OPEN SQL:

Aggregate functions:

Aggregate functions operate on multiple records to calculate one value from a group of values.

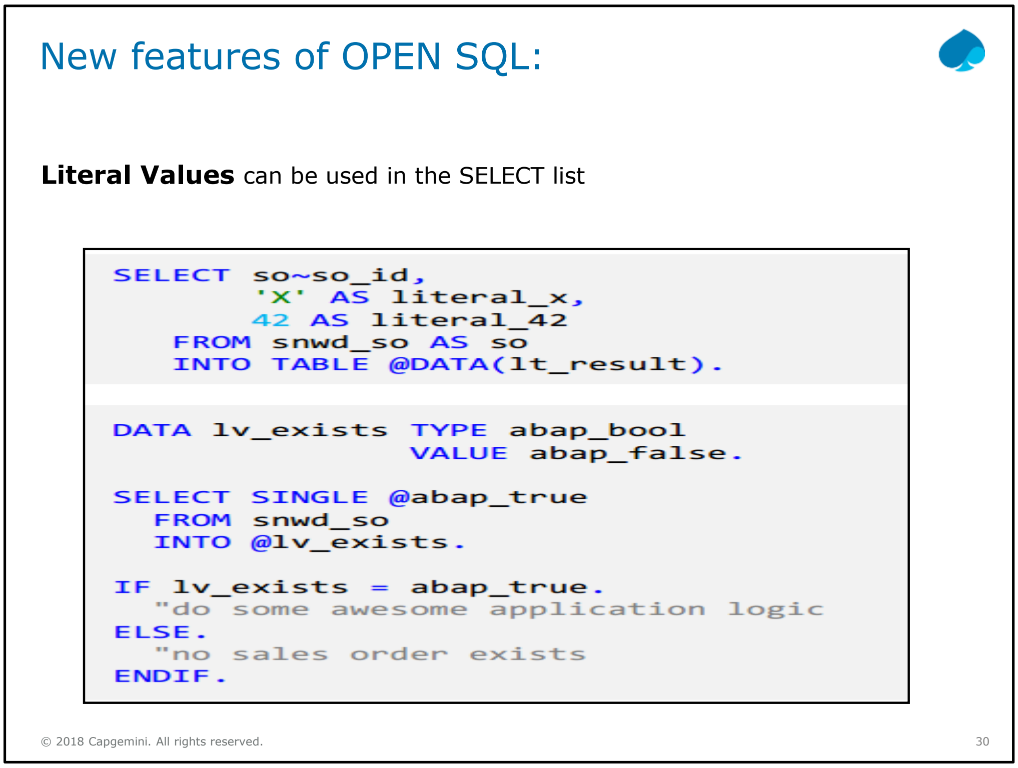
Eg. Select Sum(Sales) from table_name where Column1=’ABC’;

Sum() - returns the sum of the numeric values in a given column

Max() - returns the maximum of the numeric values in a given column

```
SELECT bp_id,
       company_name,
       so~currency_code,
       SUM( so~gross_amount )
        AS total_amount
   FROM snwd_so AS so
   INNER JOIN snwd_bpa AS bpa
   ON bpa~node_key = so~buyer_guid
   INTO TABLE @DATA(lt_result)
   WHERE so~delivery_status = ' '
   GROUP BY
     bp_id,
     company_name,
     so~currency_code
   HAVING SUM( so~gross_amount ) > 10000000.
```

29

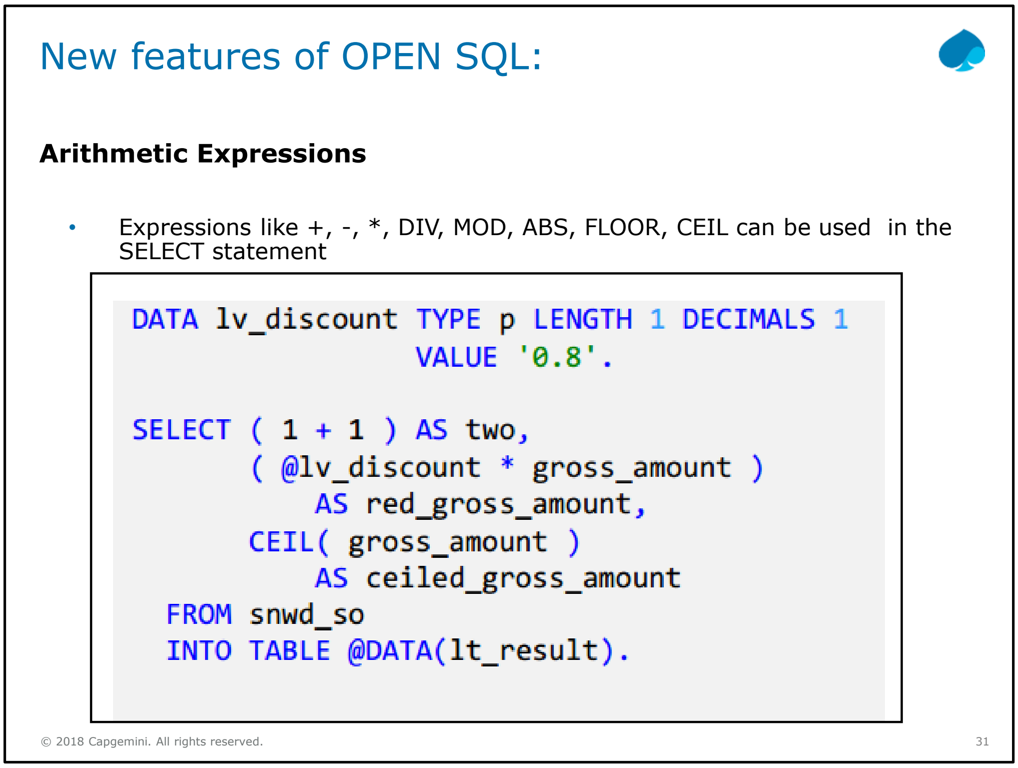


New features of OPEN SQL:

Literal Values can be used in the SELECT list

```
SELECT so~so_id,
       'X' AS literal_x,
       42 AS literal_42
   FROM snwd_so AS so
   INTO TABLE @DATA(lt_result).
```

```
DATA lv_exists TYPE abap_bool
               VALUE abap_false.

SELECT SINGLE @abap_true
  FROM snwd_so
  INTO @lv_exists.

IF lv_exists = abap_true.
  "do some awesome application logic
ELSE.
  "no sales order exists
ENDIF.
```

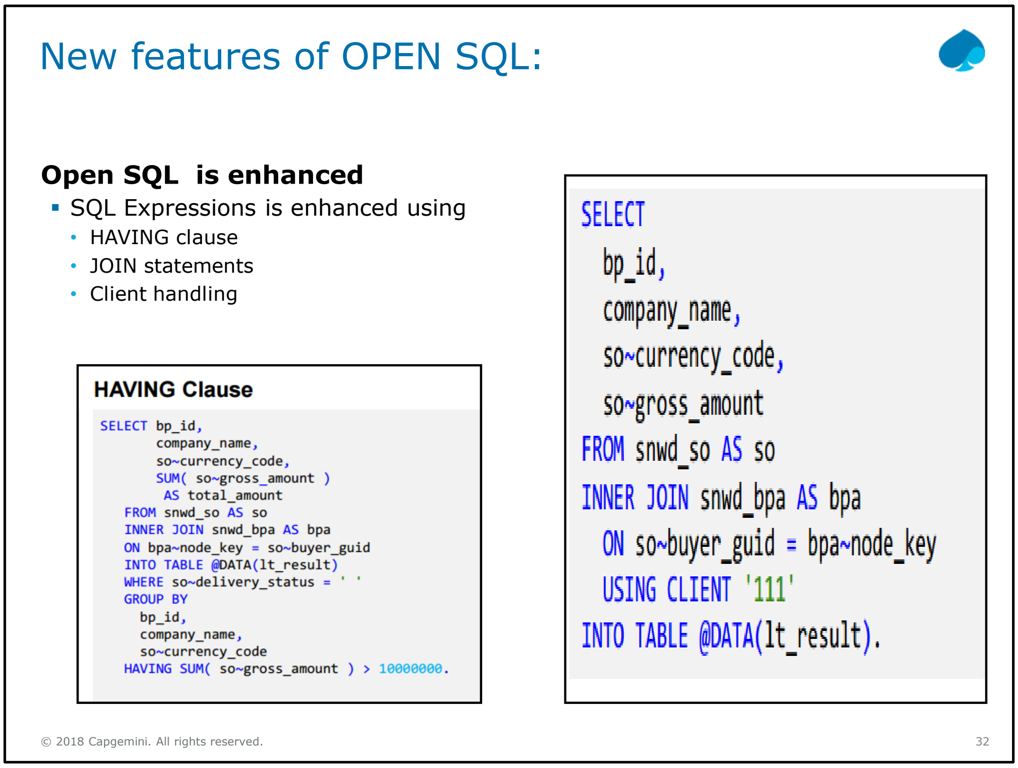


New features of OPEN SQL:

Arithmetic Expressions

- Expressions like +, -, *, DIV, MOD, ABS, FLOOR, CEIL can be used in the SELECT statement

```
DATA lv_discount TYPE p LENGTH 1 DECIMALS 1
                 VALUE '0.8'.

SELECT ( 1 + 1 ) AS two,
       ( @lv_discount * gross_amount )
           AS red_gross_amount,
       CEIL( gross_amount )
           AS ceiled_gross_amount
  FROM snwd_so
  INTO TABLE @DATA(lt_result).
```



New features of OPEN SQL:

Open SQL is enhanced

- SQL Expressions is enhanced using
  - HAVING clause
  - JOIN statements
  - Client handling

HAVING Clause

```
SELECT bp_id,
       company_name,
       so~currency_code,
       SUM( so~gross_amount )
        AS total_amount
   FROM snwd_so AS so
   INNER JOIN snwd_bpa AS bpa
   ON bpa~node_key = so~buyer_guid
   INTO TABLE @DATA(lt_result)
   WHERE so~delivery_status = ' '
   GROUP BY
     bp_id,
     company_name,
     so~currency_code
   HAVING SUM( so~gross_amount ) > 10000000.
```

```
SELECT
  bp_id,
  company_name,
  so~currency_code,
  so~gross_amount
FROM snwd_so AS so
INNER JOIN snwd_bpa AS bpa
  ON so~buyer_guid = bpa~node_key
  USING CLIENT '111'
INTO TABLE @DATA(lt_result).
```

Demo

Program on using Select statement with comma separated fields and using host variables

33

Demo

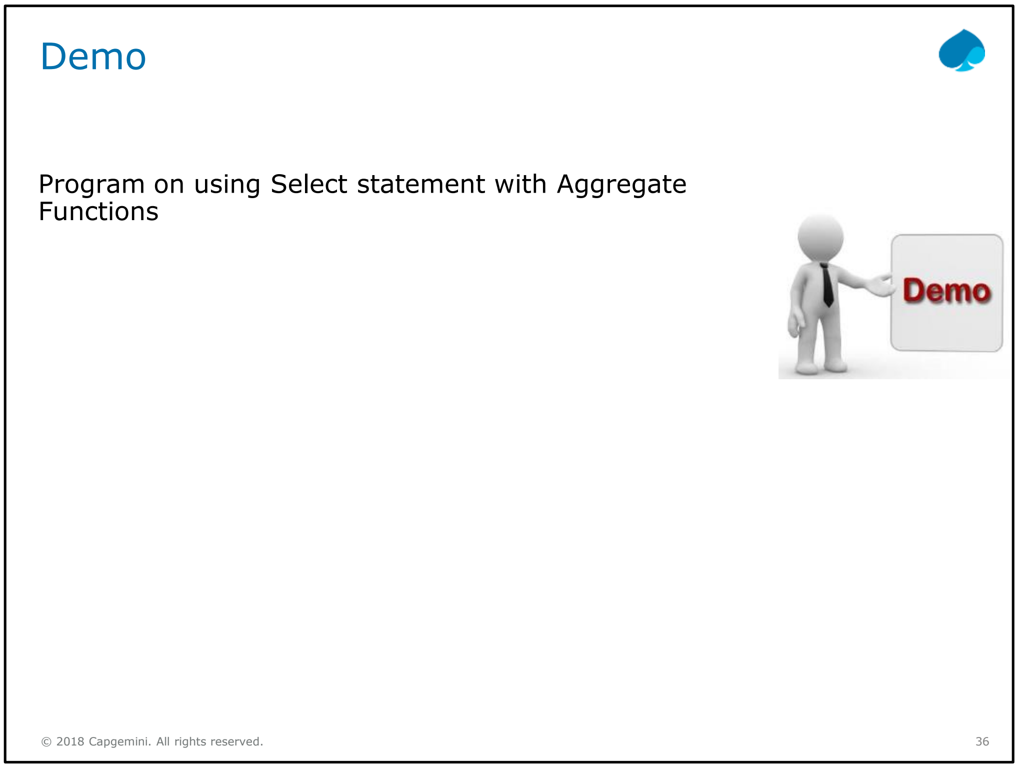
Program on using Select statement with Case Expressions

34

Demo

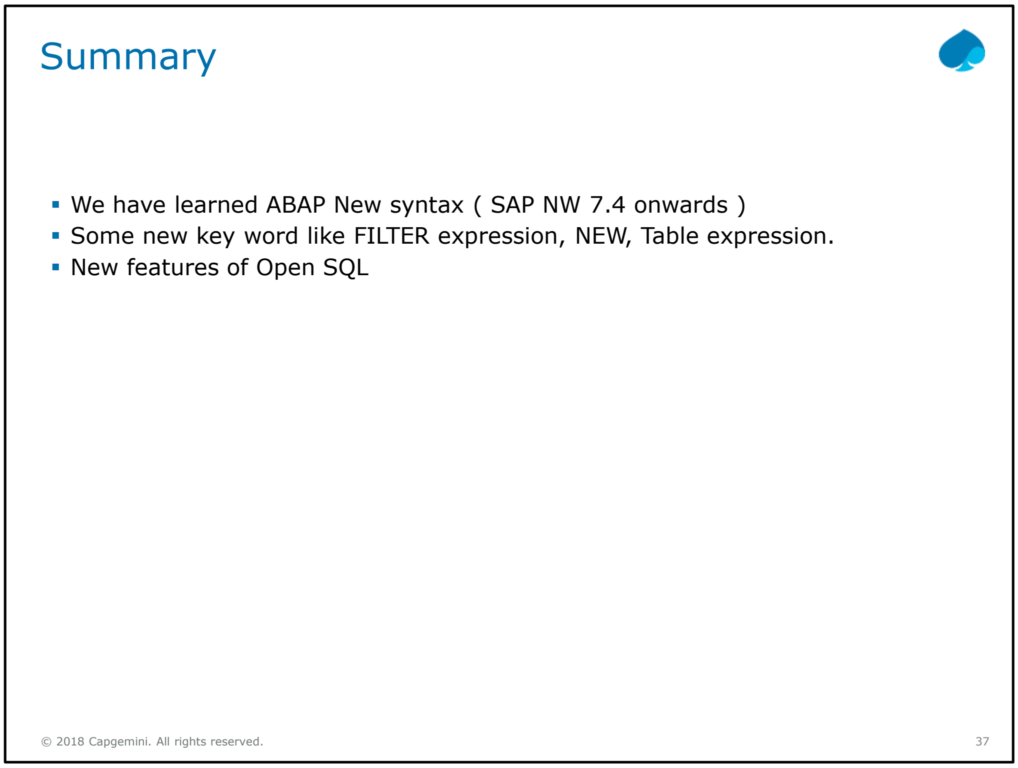
Program on using Select statement with Arithmetic Expressions

35



Demo

Program on using Select statement with Aggregate Functions



Summary

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

37