Exercise :-

Create a Program using Table SFLIGHT, SCARR, SBOOK. & Utilize Below Syntax as per requirement.

**1. Inline Declarations**

|  |  |  |
| --- | --- | --- |
| **Description** | **Before 7.40** | **With 7.40** |
| **Data statement** | **DATA text TYPE string. text = `ABC`.** | **DATA(text) = `ABC`.** |
| **Loop at into work area** | **DATA wa like LINE OF itab. LOOP AT itab INTO wa.     … ENDLOOP.** | **LOOP AT itab INTO DATA(wa).     … ENDLOOP.** |
| **Call method** | **DATA a1 TYPE …**  **DATA a2 TYPE …**  **oref->meth( IMPORTING p1 = a1**  **IMPORTING p2 = a2**  **).** | **oref->meth(**  **IMPORTING p1 = DATA(a1)**  **IMPORTING p2 = DATA(a2) ).** |
| **Loop at assigning** | **FIELD-SYMBOLS: <line> type …**  **LOOP AT itab ASSIGNING <line>.**  **…**  **ENDLOOP.** | **LOOP AT itab**  **ASSIGNING FIELD-SYMBOL(<line>).    … ENDLOOP.** |
| **Read assigning** | **FIELD-SYMBOLS: <line> type …**  **READ TABLE itab**  **ASSIGNING <line>.** | **READ TABLE itab**  **ASSIGNING FIELD-SYMBOL(<line>).** |
| **Select into**  **table** | **DATA itab TYPE TABLE OF dbtab.**  **SELECT \* FROM dbtab**  **INTO TABLE itab**  **WHERE fld1 = lv\_fld1.** | **SELECT \* FROM dbtab**  **INTO TABLE DATA(itab)**  **WHERE fld1 = @lv\_fld1.** |
| **Select single**  **into** | **SELECT SINGLE f1 f2**  **FROM dbtab**  **INTO (lv\_f1, lv\_f2)**  **WHERE …**  **WRITE: / lv\_f1, lv\_f2.** | **SELECT SINGLE f1 AS my\_f1,**  **F2 AS abc**  **FROM dbtab**  **INTO DATA(ls\_structure)**  **WHERE …**  **WRITE: / ls\_structure-my\_f1,              ls\_structure-abc.** |

**2.**[**Table Expressions**](https://blogs.sap.com/?p=86110)

If a table line is not found, the exception CX\_SY\_ITAB\_LINE\_NOT\_FOUND is raised. No sy-subrc.

|  |  |  |
| --- | --- | --- |
| **Description** | **Before 7.40** | **With 7.40** |
| **Read Table  index** | **READ TABLE itab INDEX idx**  **INTO wa.** | **wa = itab[ idx ].** |
| **Read Table  using key** | **READ TABLE itab INDEX idx**  **USING KEY key**  **INTO wa.** | **wa = itab[ KEY key INDEX idx ].** |
| **Read Table  with key** | **READ TABLE itab**  **WITH KEY col1 = …**  **col2 = …**  **INTO wa.** | **wa = itab[ col1 = … col2 = … ].** |
| **Read Table  with key components** | **READ TABLE itab**  **WITH TABLE KEY key**  **COMPONENTS col1 = …**  **col2 = …**  **INTO wa.** | **wa = itab[ KEY key col1 = …**  **col2 = … ].** |
| **Does record exist?** | **READ TABLE itab …**  **TRANSPORTING NO FIELDS.**  **IF sy-subrc = 0.**  **…**  **ENDIF.** | **IF line\_exists( itab[ … ] ).**  **…**  **ENDIF.** |
| **Get table index** | **DATA idx type sy-tabix.**  **READ TABLE …**  **TRANSPORTING NO FIELDS.**  **idx = sy-tabix.** | **DATA(idx) =**  **line\_index( itab[ … ] ).** |

**NB**: There will be a short dump if you use an inline expression that references a non-existent record.

        SAP says you should therefore assign a field symbol and check sy-subrc.

ASSIGN lt\_tab[ 1 ] to FIELD–SYMBOL(<ls\_tab>).  
IF sy–subrc = 0.  
…  
ENDIF.

**NB**: Use itab [ table\_line = … ] for untyped tables.

**3. Conversion Operator CONV**

**I.  Definition**

**CONV dtype|#( … )**

**dtype** = Type you want to convert to (explicit)

**#**     = compiler must use the context to decide the type to convert to (implicit)

**II. Example**

Method cl\_abap\_codepage=>convert\_to expects a string

| **Before 7.40** |
| --- |
| DATA text   TYPE c LENGTH 255.  DATA helper TYPE string.  DATA xstr   TYPE xstring.  helper = text.  xstr = cl\_abap\_codepage=>convert\_to( source = helper ). |
| **With 7.40** |
| DATA text TYPE c LENGTH 255.  DATA(xstr) = cl\_abap\_codepage=>convert\_to( source = CONV string( text ) ).  OR  DATA(xstr) = cl\_abap\_codepage=>convert\_to( source = CONV #( text ) ). |

**4. Value Operator VALUE**

**I. Definition**

**Variables:**    VALUE dtype|#( )

**Structures:**  VALUE dtype|#( comp1 = a1 comp2 = a2 … )

**Tables:**        VALUE dtype|#( ( … ) ( … ) … ) …

**II. Example for structures**

     TYPES:  BEGIN OF ty\_columns1, “Simple structure  
                     cols1 TYPE i,  
                     cols2 TYPE i,  
                   END OF ty\_columns1.

      TYPES: BEGIN OF ty\_columnns2,  “Nested structure  
                     coln1 TYPE i,  
                     coln2 TYPE ty\_columns1,  
                  END OF ty\_columns2.

      DATA: struc\_simple TYPE ty\_columns1,  
                struc\_nest    TYPE ty\_columns2.

     struct\_nest   = VALUE t\_struct(coln1 = 1  
                                                  coln2-cols1 = 1  
                                                  coln2-cols2 = 2 ).

**OR**

     struct\_nest   = VALUE t\_struct(coln1 = 1  
                                                   coln2 = VALUE #( cols1 = 1  
                                                   cols2 = 2 ) ).

**III. Examples for internal tables**

Elementary line type:

TYPES t\_itab TYPE TABLE OF i WITH EMPTY KEY.

DATA itab TYPE t\_itab.

itab = VALUE #( ( ) ( 1 ) ( 2 ) ).

Structured line type (RANGES table):

DATA itab TYPE RANGE OF i.

itab = VALUE #( sign = ‘I’  option = ‘BT’ ( low = 1  high = 10 )  
( low = 21 high = 30 )  
( low = 41 high = 50 )  
option = ‘GE’ ( low = 61 )  ).

**5. FOR operator**

**I. Definition**

     FOR wa|<fs> IN itab [INDEX INTO idx] [cond]

**II. Explanation**

This effectively causes a loop at itab. For each loop the row read is assigned to a work area (wa) or field-symbol(<fs>).

This wa or <fs> is local to the expression i.e. if declared in a subrourine the variable wa or <fs> is a local variable of

that subroutine. Index like SY-TABIX in loop.

Given:

TYPES: BEGIN OF ty\_ship,  
           tknum TYPE tknum,     *“Shipment Number*  
           name  TYPE ernam,     *“Name of Person who Created the Object*  
           city  TYPE ort01,     *“Starting city*  
           route TYPE route,     *“Shipment route*  
       END OF ty\_ship.  
TYPES: ty\_ships TYPE SORTED TABLE OF ty\_ship WITH UNIQUE KEY tknum.  
TYPES: ty\_citys TYPE STANDARD TABLE OF ort01 WITH EMPTY KEY.

GT\_SHIPS type ty\_ships. -> has been populated as follows:

| Row | **TKNUM[C(10)]** | **Name[C(12)]** | City**[C(25)]** | Route[C(6)] |
| --- | --- | --- | --- | --- |
| 1 | 001 | John | Melbourne | R0001 |
| 2 | 002 | Gavin | Sydney | R0003 |
| 3 | 003 | Lucy | Adelaide | R0001 |
| 4 | 004 | Elaine | Perth | R0003 |

**III. Example 1**

Populate internal table GT\_CITYS with the cities from GT\_SHIPS.

| **Before 7.40** |
| --- |
| DATA: gt\_citys TYPE ty\_citys,        gs\_ship  TYPE ty\_ship,        gs\_city  TYPE ort01.  LOOP AT gt\_ships INTO gs\_ship.   gs\_city =  gs\_ship–city.   APPEND gs\_city TO gt\_citys. ENDLOOP. |
| **With 7.40** |
| DATA(gt\_citys) = VALUE ty\_citys( FOR ls\_ship IN gt\_ships ( ls\_ship–city ) ). |

**IV. Example 2**

Populate internal table GT\_CITYS with the cities from GT\_SHIPS where the route is R0001.

| **Before 7.40** |
| --- |
| DATA: gt\_citys TYPE ty\_citys,        gs\_ship  TYPE ty\_ship,        gs\_city  TYPE ort01.  LOOP AT gt\_ships INTO gs\_ship WHERE route = ‘R0001’.   gs\_city =  gs\_ship–city.   APPEND gs\_city TO gt\_citys. ENDLOOP. |
| **With 7.40** |
| DATA(gt\_citys) = VALUE ty\_citys( FOR ls\_ship IN gt\_ships                                 WHERE ( route = ‘R0001’ ) ( ls\_ship–city ) ). |

**Note: ls\_ship does not appear to have been declared but it is declared implicitly.**

**V. FOR with THEN and UNTIL|WHILE**

FOR i = … [THEN expr] UNTIL|WHILE log\_exp

Populate an internal table as follows:

TYPES:  
  BEGIN OF ty\_line,  
    col1 TYPE i,  
    col2 TYPE i,  
    col3 TYPE i,  
  END OF ty\_line,  
  ty\_tab TYPE STANDARD TABLE OF ty\_line WITH EMPTY KEY.

| **Before 7.40** |
| --- |
| DATA: gt\_itab TYPE ty\_tab,       j       TYPE i. FIELD-SYMBOLS <ls\_tab> TYPE ty\_line.  j = 1. DO. j = j + 10. IF j > 40. EXIT. ENDIF. APPEND INITIAL LINE TO gt\_itab ASSIGNING <ls\_tab>. <ls\_tab>–col1 = j. <ls\_tab>–col2 = j + 1. <ls\_tab>–col3 = j + 2. ENDDO. |
| **With 7.40** |
| DATA(gt\_itab) = VALUE ty\_tab( FOR j = 11 THEN j + 10 UNTIL j > 40                             ( col1 = j col2 = j + 1 col3 = j + 2  ) ). |

**6. Reduction operator REDUCE**

**I. Definition**

… REDUCE type(

INIT result = start\_value

           …

FOR for\_exp1

FOR for\_exp2

…

NEXT …

           result = iterated\_value

… )

**II. Note**

     While VALUE and NEW expressions can include FOR expressions, REDUCE must include at least one FOR expression. You can use all kinds      of FOR expressions in REDUCE:

* with IN for iterating internal tables
* with UNTIL or WHILE for conditional iterations

**III. Example 1**

Count lines of table that meet a condition (field F1 contains “XYZ”).

| **Before 7.40** |
| --- |
| DATA: lv\_lines TYPE i.  LOOP AT gt\_itab INTO ls\_itab where F1 = ‘XYZ’.   lv\_lines = lv\_lines + 1. ENDLOOP. |
| **With 7.40** |
| DATA(lv\_lines) = REDUCE i( INIT x = 0 FOR wa IN gt\_itab                      WHERE( F1 = ‘XYZ’ ) NEXT x = x + 1 ). |

**IV. Example 2**

Sum the values 1 to 10 stored in the column of a table defined as follows

DATA gt\_itab TYPE STANDARD TABLE OF i WITH EMPTY KEY.  
gt\_itab = VALUE #( FOR j = 1 WHILE j <= 10 ( j ) ).

| **Before 7.40** |
| --- |
| DATA: lv\_line TYPE i,       lv\_sum  TYPE i.  LOOP AT gt\_itab INTO lv\_line.   lv\_sum = lv\_sum + lv\_line. ENDLOOP. |
| **With 7.40** |
| DATA(lv\_sum) = REDUCE i( INIT x = 0 FOR wa IN itab NEXT x = x + wa ). |

**V. Example 3**

Using a class reference – works because “write” method returns reference to instance object

| **With 7.40** |
| --- |
| TYPES outref TYPE REF TO if\_demo\_output.  DATA(output) = REDUCE outref( INIT out  = cl\_demo\_output=>new( )                               text = `Count up:`                               FOR n = 1 UNTIL n > 11                               NEXT out = out->write( text )                               text = |{ n }| ).  output->display( ). |

**7. Conditional operators COND and SWITCH**

**I. Definition**

… COND dtype|#( WHEN log\_exp1 THEN result1  
[ WHEN log\_exp2 THEN result2 ]  
…  
[ ELSE resultn ] ) …

… SWITCH dtype|#( operand  
WHEN const1 THEN result1  
[ WHEN const2 THEN result2 ]  
…  
[ ELSE resultn ] ) …

**II. Example for COND**

DATA(time) =

  COND string(

    WHEN sy-timlo < ‘120000’ THEN

      |{ sy-timlo TIME = ISO } AM|

    WHEN sy-timlo > ‘120000’ THEN

      |{ CONV t( sy-timlo – 12 \* 3600 )

TIME = ISO } PM|

    WHEN sy-timlo = ‘120000’ THEN

      |High Noon|

    ELSE

      THROW cx\_cant\_be( ) ).

**III. Example for SWITCH**

DATA(text) =  
NEW class( )->meth(  
                     SWITCH #( sy-langu  
                              WHEN ‘D’ THEN `DE`  
                              WHEN ‘E’ THEN `EN`  
                               ELSE THROW cx\_langu\_not\_supported( ) ) ).

**8. Corresponding Operator**

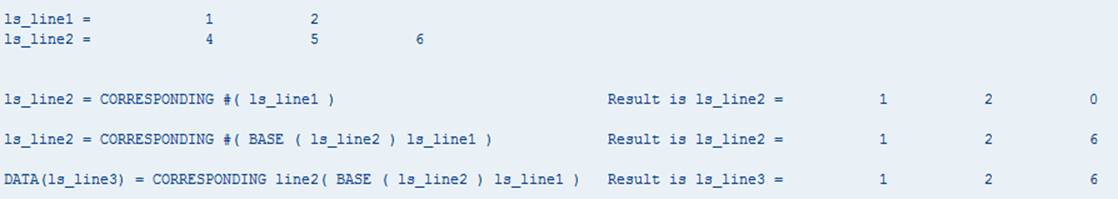
**I. Definition**

**… CORRESPONDING type( [BASE ( base )] struct|itab [mapping|except] )**

**II. Example Code**

| **With 7.40** |
| --- |
| TYPES: BEGIN OF line1, col1 TYPE i, col2 TYPE i, END OF line1. TYPES: BEGIN OF line2, col1 TYPE i, col2 TYPE i, col3 TYPE i, END OF line2.  DATA(ls\_line1) = VALUE line1( col1 = 1 col2 = 2 ). WRITE: / ‘ls\_line1 =’ ,15 ls\_line1–col1, ls\_line1–col2. DATA(ls\_line2) = VALUE line2( col1 = 4 col2 = 5 col3 = 6 ). WRITE: / ‘ls\_line2 =’ ,15 ls\_line2–col1, ls\_line2–col2, ls\_line2–col3. SKIP 2.  ls\_line2 = CORRESPONDING #( ls\_line1 ). WRITE: / ‘ls\_line2 = CORRESPONDING #( ls\_line1 )’       ,70 ‘Result is ls\_line2 = ‘          ,ls\_line2–col1, ls\_line2–col2, ls\_line2–col3. SKIP.  ls\_line2 = VALUE line2( col1 = 4 col2 = 5 col3 = 6 ).   *“Restore ls\_line2* ls\_line2 = CORRESPONDING #( BASE ( ls\_line2 ) ls\_line1 ). WRITE: / ‘ls\_line2 = CORRESPONDING #( BASE ( ls\_line2 ) ls\_line1 )’          , 70 ‘Result is ls\_line2 = ‘, ls\_line2–col1          , ls\_line2–col2, ls\_line2–col3. SKIP.  ls\_line2 = VALUE line2( col1 = 4 col2 = 5 col3 = 6 ).   *“Restore ls\_line2* DATA(ls\_line3) = CORRESPONDING line2( BASE ( ls\_line2 ) ls\_line1 ). WRITE: / ‘DATA(ls\_line3) = CORRESPONDING line2( BASE ( ls\_line2 ) ls\_line1 )’           , 70 ‘Result is ls\_line3 = ‘ , ls\_line3–col1           , ls\_line3–col2, ls\_line3–col3. |

**III. Output**

[](https://blogs.sap.com/wp-content/uploads/2015/10/image001_906951.jpg)

**IV. Explanation**

Given structures ls\_line1 & ls\_line2 defined and populated as above.

|  | **Before 7.40** | **With 7.40** |
| --- | --- | --- |
| **1** | **CLEAR ls\_line2.**  **MOVE-CORRESPONDING ls\_line1**  **TO ls\_line2.** | **ls\_line2 = CORRESPONDING #( ls\_line1 ).** |
| **2** | **MOVE-CORRESPONDING ls\_line1**  **TO ls\_line2.** | **ls\_line2 = CORRESPONDING #**  **( BASE ( ls\_line2 ) ls\_line1 ).** |
| **3** | **DATA: ls\_line3 like ls\_line2.**  **ls\_line3 = ls\_line2.**  **MOVE-CORRESPONDING ls\_line1**  **TO ls\_line2.** | **DATA(ls\_line3) = CORRESPONDING line2**  **( BASE ( ls\_line2 ) ls\_line1 ).** |

1. The contents of ls\_line1 are moved to ls\_line2 where there is a matching column name. Where there is no

            match the column of ls\_line2 **is initialised.**

  2. This uses the existing contents of ls\_line2 as a base and overwrites the matching columns from ls\_line1.

**This is exactly like MOVE-CORRESPONDING.**

  3. This creates a third and new structure (ls\_line3) which is based on ls\_line2 but overwritten by matching

             columns of ls\_line1.

**V. Additions MAPPING and EXCEPT**

   MAPPING allows you to map fields with non-identically named components to qualify for the data transfer.

**… MAPPING  t1 = s1 t2 = s2**

   EXCEPT allows you to list fields that must be excluded from the data transfer

**… EXCEPT  {t1 t2 …}**

**9. Strings**

**I. String Templates**

A string template is enclosed by two characters “**|**” and creates a character string.

Literal text consists of all characters that are not in braces {}. The braces can contain:

* data objects,
* calculation expressions,
* constructor expressions,
* table expressions,
* predefined functions, or
* functional methods and method chainings

| **Before 7.40** |
| --- |
| DATA itab TYPE TABLE OF scarr.  SELECT \* FROM scarr INTO TABLE itab.  DATA wa LIKE LINE OF itab.  READ TABLE itab WITH KEY carrid = ‘LH’ INTO wa.  DATA output TYPE string.  CONCATENATE ‘Carrier:’ wa-carrname INTO output SEPARATED BY space.  cl\_demo\_output=>display( output ). |
| **With 7.40** |
| SELECT \* FROM scarr INTO TABLE @DATA(lt\_scarr). cl\_demo\_output=>display( |Carrier: { lt\_scarr[ carrid = ‘LH’ ]–carrname }|  ). |

**II. Concatenation**

| **Before 7.40** |
| --- |
| DATA lv\_output TYPE string. CONCATENATE ‘Hello’ ‘world’ INTO lv\_output SEPARATED BY space. |
| **With 7.40** |
| DATA(lv\_out) = |Hello| & | | & |world|. |

**III. Width/Alignment/Padding**

WRITE / |{ ‘Left’     WIDTH = 20 ALIGN = LEFT   PAD = ‘0’ }|.  
WRITE / |{ ‘Centre’   WIDTH = 20 ALIGN = CENTER PAD = ‘0’ }|.  
WRITE / |{ ‘Right’    WIDTH = 20 ALIGN = RIGHT  PAD = ‘0’ }|.

**IV. Case**

WRITE / |{ ‘Text’ CASE = (cl\_abap\_format=>c\_raw) }|.  
WRITE / |{ ‘Text’ CASE = (cl\_abap\_format=>c\_upper) }|.  
WRITE / |{ ‘Text’ CASE = (cl\_abap\_format=>c\_lower) }|.

**V. ALPHA conversion**

DATA(lv\_vbeln) = ‘0000012345’.  
WRITE / |{ lv\_vbeln  ALPHA = OUT }|.     “or use ALPHA = IN to go in other direction

**VI. Date conversion**

WRITE / |{ pa\_date DATE = ISO }|.           “Date Format YYYY-MM-DD  
WRITE / |{ pa\_date DATE = User }|.          “As per user settings  
WRITE / |{ pa\_date DATE = Environment }|.   “Formatting setting of language environment

**10. Loop at Group By**

**I. Definition**

**LOOP AT itab result [cond] GROUP BY key ( key1 = dobj1 key2 = dobj2 …  
      [gs = GROUP SIZE] [gi = GROUP INDEX] )  
      [ASCENDING|DESCENDING [AS TEXT]]  
      [WITHOUT MEMBERS]  
      [{INTO group}|{ASSIGNING <group>}]  
      …  
      [LOOP AT GROUP group|<group>  
      …  
      ENDLOOP.]  
      …**

**ENDLOOP.**

**II. Explanation**

The outer loop will do one iteration per key. So if 3 records match the key there will only be one iteration for these 3 records. The structure “group” (or

“<group>” ) is unusual in that it can be looped over using the “LOOP AT GROUP” statement. This will loop over the 3 records (members) of the group. The

structure “group” also contains the current key as well as the size of the group and index of the group ( if GROUP SIZE and GROUP INDEX have been

assigned a field name). This is best understood by an example.

**III. Example**

| With 7.40 |
| --- |
| TYPES: BEGIN OF ty\_employee,    name TYPE char30,    role    TYPE char30,    age    TYPE i,  END OF ty\_employee,  ty\_employee\_t TYPE STANDARD TABLE OF ty\_employee WITH KEY name.  DATA(gt\_employee) = VALUE ty\_employee\_t(  ( name = ‘John‘     role = ‘ABAP guru‘       age = 34 )  ( name = ‘Alice‘     role = ‘FI Consultant‘   age = 42 )  ( name = ‘Barry‘    role = ‘ABAP guru‘       age = 54 )  ( name = ‘Mary‘     role = ‘FI Consultant‘   age = 37 )  ( name = ‘Arthur‘   role = ‘ABAP guru‘       age = 34 )  ( name = ‘Mandy‘  role = ‘SD Consultant‘  age = 64 ) ).  DATA: gv\_tot\_age TYPE i,             gv\_avg\_age TYPE decfloat34.  “Loop with grouping on Role  LOOP AT gt\_employee INTO DATA(ls\_employee)    GROUP BY ( role  = ls\_employee-role                          size  = GROUP SIZE                         index = GROUP INDEX )    ASCENDING    ASSIGNING FIELD-SYMBOL(<group>).    CLEAR: gv\_tot\_age.    “Output info at group level    WRITE: / |Group: { <group>-index }    Role: { <group>-role WIDTH = 15 }|                & |     Number in this role: { <group>-size }|.     “Loop at members of the group     LOOP AT GROUP <group> ASSIGNING FIELD-SYMBOL(<ls\_member>).        gv\_tot\_age = gv\_tot\_age + <ls\_member>-age.        WRITE: /13 <ls\_member>-name.     ENDLOOP.     “Average age     gv\_avg\_age = gv\_tot\_age / <group>-size.     WRITE: / |Average age: { gv\_avg\_age }|.     SKIP.  ENDLOOP. |

**IV. Output**

Group: 1    Role: ABAP guru           Number in this role: 3

                 John

                 Barry

                 Arthur

Average age: 40.66666666666666666666666666666667

Group: 2    Role: FI Consultant       Number in this role: 2

                  Alice

                  Mary

Average age: 39.5

Group: 3    Role: SD Consultant       Number in this role: 1

                  Mandy

Average age: 64

**11. Classes/Methods**

**I. Referencing fields within returned structures**

| **Before 7.40** |
| --- |
| DATA: ls\_lfa1  TYPE lfa1,       lv\_name1 TYPE lfa1–name1.  ls\_lfa1  = My\_Class=>get\_lfa1( ). lv\_name1 = ls\_lfa1–name1. |
| **With 7.40** |
| DATA(lv\_name1) = My\_Class=>get\_lfa1( )–name1. |

**II. Methods that return a type BOOLEAN**

| **Before 7.40** |
| --- |
| IF My\_Class=>return\_boolean( ) = abap\_true.  …  ENDIF. |
| **With 7.40** |
| IF My\_Class=>return\_boolean( ).  …  ENDIF. |

NB: The type “BOOLEAN” is not a true Boolean but a char1 with allowed values X,- and <blank>.

       Using type “FLAG” or “WDY\_BOOLEAN” works just as well.

**III. NEW operator**

This operator can be used to instantiate an object.

| **Before 7.40** |
| --- |
| DATA: lo\_delivs TYPE REF TO zcl\_sd\_delivs,              lo\_deliv  TYPE REF TO zcl\_sd\_deliv.  CREATE OBJECT lo\_delivs. CREATE OBJECT lo\_deliv.  lo\_deliv = lo\_delivs->get\_deliv( lv\_vbeln ). |
| **With 7.40** |
| DATA(lo\_deliv) = new zcl\_sd\_delivs( )->get\_deliv( lv\_vbeln ). |

**12. Meshes**

Allows an association to be set up between related data groups.

**I. Problem**

Given the following 2 internal tables:

TYPES: BEGIN OF t\_manager,  
name   TYPE char10,  
salary TYPE int4,  
END OF t\_manager,  
tt\_manager TYPE SORTED TABLE OF t\_manager WITH UNIQUE KEY name.

TYPES: BEGIN OF t\_developer,  
name    TYPE char10,  
salary  TYPE int4,  
manager TYPE char10,   *“Name of manager*  
END OF t\_developer,  
tt\_developer TYPE SORTED TABLE OF t\_developer WITH UNIQUE KEY name.

**Populated as follows:**

| Row | **Name[C(10)]** | Salary**[I(4)]** |
| --- | --- | --- |
| 1 | Jason | 3000 |
| 2 | Thomas | 3200 |

| Row | | **Name[C(10)]** | | --- | | Salary**[I(4)** | Manager**[C(10)]** |
| --- | --- | --- | --- | --- |
| 1 | Bob | 2100 | Jason |
| 2 | David | 2000 | Thomas |
| 3 | Jack | 1000 | Thomas |
| 4 | Jerry | 1000 | Jason |
| 5 | John | 2100 | Thomas |
| 6 | Tom | 2000 | Jason |

Get the details of Jerry’s manager and all developers managed by Thomas.

**II. Solution**

| **With 7.40** |
| --- |
| TYPES: BEGIN OF MESH m\_team,          managers   TYPE tt\_manager  ASSOCIATION my\_employee TO developers                                                              ON manager = name,          developers TYPE tt\_developer ASSOCIATION my\_manager TO managers                                                              ON name = manager,        END OF MESH m\_team.  DATA: ls\_team TYPE m\_team. ls\_team–managers   = lt\_manager. ls\_team–developers = lt\_developer.  *\*Get details of Jerry’s manager \**  *“get line of dev table*  ASSIGN lt\_developer[ name = ‘Jerry’ ] TO FIELD–SYMBOL(<ls\_jerry>). DATA(ls\_jmanager) =  ls\_team–developers\my\_manager[ <ls\_jerry> ].  WRITE: / |Jerry‘s manager: { ls\_jmanager-name }|,30                    |Salary: { ls\_jmanager-salary }|.  *“Get Thomas’ developers* SKIP. WRITE: / |Thomas‘ developers:|.  *“line of manager table*  ASSIGN lt\_manager[ name = ‘Thomas’ ] TO FIELD–SYMBOL(<ls\_thomas>). LOOP AT ls\_team–managers\my\_employee[ <ls\_thomas> ]          ASSIGNING FIELD–SYMBOL(<ls\_emp>).    WRITE: / |Employee name: { <ls\_emp>–name }|. ENDLOOP. |

**III. Output**

     Jerry’s manager: Jason          Salary: 3000

     Thomas’ developers:

     Employee name: David

     Employee name: Jack

     Employee name: John

**13. Filter**

Filter the records in a table based on records in another table.

**I. Definition**

… FILTER type( itab [EXCEPT] [IN ftab] [USING KEY keyname]  
           WHERE c1 op f1 [AND c2 op f2 […]] )

**II. Problem**

Filter an internal table of Flight Schedules (SPFLI) to only those flights based on a filter table that contains the fields Cityfrom and CityTo.

**III. Solution**

| **With 7.40** |
| --- |
| TYPES: BEGIN OF ty\_filter,          cityfrom TYPE spfli–cityfrom,          cityto   TYPE spfli–cityto,          f3       TYPE i,        END OF ty\_filter,        ty\_filter\_tab TYPE HASHED TABLE OF ty\_filter                       WITH UNIQUE KEY cityfrom cityto. DATA: lt\_splfi TYPE STANDARD TABLE OF spfli.  SELECT \* FROM spfli APPENDING TABLE lt\_splfi.  DATA(lt\_filter) = VALUE ty\_filter\_tab( f3 = 2                            ( cityfrom = ‘NEW YORK’  cityto  = ‘SAN FRANCISCO’ )              ( cityfrom = ‘FRANKFURT’ cityto  = ‘NEW YORK’ )  ).  DATA(lt\_myrecs) = FILTER #( lt\_splfi IN lt\_filter                                    WHERE cityfrom = cityfrom                                      AND cityto = cityto ).  “Output filtered records LOOP AT lt\_myrecs ASSIGNING FIELD–SYMBOL(<ls\_rec>).   WRITE: / <ls\_rec>–carrid,8 <ls\_rec>–cityfrom,30             <ls\_rec>–cityto,45 <ls\_rec>–deptime.  ENDLOOP. |

Note: using the keyword “EXCEPT” (see definition above) would have returned the exact opposite records i.e all records EXCEPT for those those returned above.