Class: ZCL_ATABLE_HELPER

Status: Active

Attributes

Description: ATABLE AUTOMATED PROCESSING
Instantiation: Public
Final

Not released
Fixed pt.arithmetic
Program status: Customer Production Program
Category: General Object Type
Package: ZABAP
Original lang.: EN

Created by: SKALVI
Created on: 01/26/2010
Last changed by: SKALVI
Last changed on: 01/27/2010

Documentation

Functionality

This class builds the A_TABLE information required for pricing and the freight surcharges. It will work on A_TABLES with up to 13 key fields.

- The constructor uses the associated class **ZCL_ATABLE_OBJECT** when calling the CREATE OBJECT create_atable.

Methods:

CONSTRUCTOR: Creates the required A_TABLES.

Parameters:

A_TABLE_MAIN: This contains all of the fields used to determine the price or freight surcharge. This could be your primary internal table, For example: GT_INT_TABLE.

- A field in the table **must** be populated with a date, for comparison with the effective and ending pricing dates / freight charges (DATAB and DATBI).

GT_MAPPING: This is a table that contains two elements. The first element is the access sequence. These are used in the table T682I to find out which A_TABLES to use. The second element is the mapping order. This is used to determine which conditions extract from the A_TABLES and in which in order to traverse the conditions.

- For example: [{ZMAT,ZMAT}, {ZM07, ZMSD}, {ZM07, ZMVD}, {ZNET, ZNET}]

RANGE_TYPE: 0=Standard, 1=Effective, 2=Ending

P_VKORG: (Optional) Sales Organization from Selection Screen. This is used to obtain the pricing procedure from T683V, and subsequently the condition types for that pricing procedure from the custom table ZTC_PRICPRO_CTYP, and the corresponding access sequences in T685 for the condtion types selected from ZTC_PRICPRO_CTYP.

P_VTWEG: (Optional) Distribution Channel from Selection Screen. This is used to obtain the pricing procedure from

T683V, and subsequently the condition types for that pricing procedure from the custom table ZTC_PRICPRO_CTYP, and the corresponding access sequences in T685 for the condtion types selected from ZTC_PRICPRO_CTYP.

P_SPART: (Optional) Division from Selection Screen. This is used to obtain the pricing procedure from T683V, and subsequently the condition types for that pricing procedure from the custom table ZTC_PRICPRO_CTYP, and the corresponding access sequences in T685 for the condtion types selected from ZTC_PRICPRO_CTYP.

P_PARKEY: (Optional) ATABLES using partial keys. This is used if you load information from an ATABLE, where the main internal table does not have all of the key fields for a given ATABLE. For example, if a program is required to load customer and material information from A005 and KONP, up front, and the main internal table only has the customer field, you would need to use a partial key. It is strongly recommended that you use a different **instance** for partial key processing. See the program **ZSISRPT010** for a more detailed example.

GET_KONP_INFO: Gets the pricing information.

Parameters:

WORKAREA: This is the arbitrary work area based on the main internal table.

GT_MAPPING: This is a table that contains two elements. One for the ATABLE name, and the other for extracting/traversal order.

- The first element is the access sequence. These are used in the table T682I to find out which A_TABLES to use. The second element is the mapping order. This is used to determine which conditions extract from the A_TABLES and in which in order to traverse the conditions.

GT_KONP_INFO: Table returning the pricing information.

GT_ATABLE_PTR: Returns the contents of an ATABLE. See the program **ZSISRPT010** for a more detailed example.

GT_KONP_PTR: Returns the contents of the KONP table corresponding the ATABLE listed in the previous parameter. See the program **ZSISRPT010** for a more detailed example.

Here is an Example of how to use this class in a program:

```
*&---------------------------------------------------------------------*
*& Report  Z_ATABLE_DEMO
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*

REPORT  z_atable_demo.
*---------------------------------------------------------------------*
*  Arbitrary Structure with fields interspersed for A_TABLE access    *
*---------------------------------------------------------------------*
TYPES: BEGIN OF test1,
          vkorg  TYPE a005-vkorg,
          kunnr  TYPE kna1-kunnr,
          altkn  TYPE knb1-altkn,
          name1  TYPE kna1-name1,
          text1  TYPE t052u-text1,
          bonus  TYPE bonus,
          vkgrp  TYPE knvv-vkgrp,
          vkbur  TYPE knvv-vkbur,
          vwerk  TYPE knvv-vwerk,
          spart  TYPE spart,
          kvgr2  TYPE knvv-kvgr2,
          kdgrp  TYPE knvv-kdgrp,
          konda  TYPE knvv-konda,
          matnr  TYPE mara-matnr,
```

```
          bismt  TYPE mara-bismt,
          postx  TYPE knmt-postx,
          matkl  TYPE mara-matkl,
          vtweg  TYPE a005-vtweg,
          kunag  TYPE kunag,
          mydate TYPE sy-datum,
        END OF test1.

DATA: gt_test1 TYPE STANDARD TABLE OF test1,
      wa_test1 LIKE LINE OF gt_test1,
      range_type TYPE i.
*---------------------------------------------------------------------*
*  Example 1                                                          *
*---------------------------------------------------------------------*
CLEAR wa_test1.
MOVE '2000' TO wa_test1-vkorg.       "Sales Org
MOVE '02' TO wa_test1-vtweg.         "Distribution Channel
MOVE '99' TO wa_test1-spart.         "Division
MOVE '0001000005' TO wa_test1-kunag. "Sold-To Party
MOVE '20101231' TO wa_test1-mydate.
APPEND wa_test1 TO gt_test1.
*---------------------------------------------------------------------*
*  Example 2                                                          *
*---------------------------------------------------------------------*
CLEAR wa_test1.
MOVE '2000' TO wa_test1-vkorg.       "Sales Org
MOVE '01' TO wa_test1-vtweg.         "Distribution Channel
```

```
MOVE '0001000008' TO wa_test1-kunnr.  "Customer
MOVE '000000000071000232' TO wa_test1-matnr.  "Material
MOVE '99991231' TO wa_test1-mydate.
APPEND wa_test1 TO gt_test1.
*---------------------------------------------------------------------*
*  Example 3                                                          *
*---------------------------------------------------------------------*
CLEAR wa_test1.
MOVE '2000' TO wa_test1-vkorg.       "Sales Org
MOVE '01' TO wa_test1-vtweg.         "Distribution Channel
MOVE '0001000008' TO wa_test1-kunnr.  "Customer
MOVE '000000000071000234' TO wa_test1-matnr.  "Material
MOVE '99991231' TO wa_test1-mydate.
APPEND wa_test1 TO gt_test1.
*---------------------------------------------------------------------*
*  Example 4                                                          *
*---------------------------------------------------------------------*
CLEAR wa_test1.
MOVE  '2000'  TO  wa_test1-vkorg.     "Sales  Org
MOVE  '0010000531' TO wa_test1-kunnr. "Customer
MOVE '14' TO wa_test1-bonus.  "VRG
MOVE '20101231' TO wa_test1-mydate.
APPEND wa_test1 TO gt_test1.
*---------------------------------------------------------------------*
*  Example 5                                                          *
*---------------------------------------------------------------------*
CLEAR wa_test1.
```

```
MOVE '2000' TO wa_test1-vkorg.     "Sales Org
MOVE '0010000531' TO wa_test1-kunnr. "Customer
MOVE '99' TO wa_test1-bonus.  "VRG
MOVE '20101231' TO wa_test1-mydate.

DATA: atable_helper TYPE REF TO zcl_atable_helper.
*---------------------------------------------------------------------*
*  Mapping Table Structure                                            *
*---------------------------------------------------------------------*
TYPES: BEGIN OF ty_mapping,
        acc_seq TYPE c LENGTH 4,
        map_ord TYPE c LENGTH 4,
       END OF ty_mapping.
*---------------------------------------------------------------------*
*  KONP pricing structure                                             *
*---------------------------------------------------------------------*
TYPES: BEGIN OF ty_konp_info,
                aname TYPE kschl,
                kschl TYPE kscha,
                knumh TYPE knumb,
                datbi TYPE datum,
                datab TYPE datum,
                kbetr TYPE konp-kbetr,
                konwa TYPE konp-konwa,
                kpein TYPE konp-kpein,
                kmein TYPE konp-kmein,
       END OF ty_konp_info.
```

```
*---------------------------------------------------------------------*
*  Table and Work Area Definitions                                    *
*---------------------------------------------------------------------*
DATA:  gt_mapping TYPE STANDARD TABLE OF ty_mapping.
DATA:  gt_konp_info TYPE STANDARD TABLE OF ty_konp_info.
DATA:  wa_mapping LIKE LINE OF gt_mapping.
DATA:  wa_konp_info LIKE LINE OF gt_konp_info.
*---------------------------------------------------------------------*
*  Specify access sequences to use, and the order in which they are   *
*  to be checked respectively, in the mapping table.                  *
*---------------------------------------------------------------------*
REFRESH gt_mapping.
wa_mapping-acc_seq = 'ZMAT'.
wa_mapping-map_ord = 'ZMAT'.
APPEND wa_mapping TO gt_mapping.
wa_mapping-acc_seq = 'ZM07'.
wa_mapping-map_ord = 'ZMSD'.
APPEND wa_mapping TO gt_mapping.
wa_mapping-acc_seq = 'ZM07'.
wa_mapping-map_ord = 'ZMVD'.
APPEND wa_mapping TO gt_mapping.
wa_mapping-acc_seq = 'ZNET'.
wa_mapping-map_ord = 'ZNET'.
APPEND wa_mapping TO gt_mapping.
*---------------------------------------------------------------------*
*  Call the helper class to create all of the A_TABLE objects corres- *
*  ponding to the values stored in the main internal table (gt_test1).*
```

```
*---------------------------------------------------------------------*
CREATE OBJECT atable_helper
  EXPORTING
    a_table_main = gt_test1
    gt_mapping   = gt_mapping
    range_type   = 1.
*---------------------------------------------------------------------*
*  Call the helper class to get the pricing info for the A_TABLES     *
*---------------------------------------------------------------------*
LOOP AT gt_test1 INTO wa_test1.
  CALL METHOD atable_helper->get_konp_info
    EXPORTING
      workarea     = wa_test1
      gt_mapping   = gt_mapping
    IMPORTING
      gt_konp_info = gt_konp_info.
  WRITE : /.
*---------------------------------------------------------------------*
*  Example 1                                                          *
*---------------------------------------------------------------------*
  IF sy-tabix = 1.
    WRITE : / 'Example #1'.
    WRITE : / 'Sales Org:', wa_test1-vkorg,  'Dist. Ch:', wa_test1-vtweg ,
'Division:', wa_test1-spart, 'Sold-to-pty:', wa_test1-kunag, 'Valid to',
wa_test1-mydate.
  ENDIF.
*---------------------------------------------------------------------*
*  Example 2                                                          *
```

```
*---------------------------------------------------------------------*

  IF sy-tabix = 2.

    WRITE : / 'Example #2'.

    WRITE : / 'Sales Org:', wa_test1-vkorg,  'Dist Ch:', wa_test1-vtweg,
'Customer:', wa_test1-kunnr, 'Material:', wa_test1-matnr, 'Valid to:',
wa_test1-mydate.

  ENDIF.

*---------------------------------------------------------------------*

*  Example 3                                                          *

*---------------------------------------------------------------------*

  IF sy-tabix = 3.

    WRITE : / 'Example #3'.

    WRITE : / 'Sales Org:', wa_test1-vkorg,  'Dist Ch:', wa_test1-vtweg,
'Customer:', wa_test1-kunnr, 'Material:', wa_test1-matnr ,'Valid to:',
wa_test1-mydate.

  ENDIF.

*---------------------------------------------------------------------*

*  Example 4                                                          *

*---------------------------------------------------------------------*

  IF sy-tabix = 4.

    WRITE : / 'Example #4'.

    WRITE : / 'Sales Org:', wa_test1-vkorg,  'Customer:', wa_test1-kunnr ,
'Volume Rebate Group:', wa_test1-bonus, 'Valid to:',  wa_test1-mydate.

  ENDIF.

*---------------------------------------------------------------------*

*  Example 5                                                          *

*---------------------------------------------------------------------*

  IF sy-tabix = 5.

    WRITE : / 'Example #5'.

    WRITE : / 'Sales Org:', wa_test1-vkorg,  'Customer:', wa_test1-kunnr ,
```

```
'Volume Rebate Group:', wa_test1-bonus, 'Valid to:', wa_test1-mydate.

  ENDIF.

*---------------------------------------------------------------------*

*  Write pricing conditions for any example above                     *

*---------------------------------------------------------------------*

  LOOP AT gt_konp_info INTO wa_konp_info.

    WRITE : / 'Pricing info:'.

    WRITE : / 'A_Table:', wa_konp_info-aname, 'Condition:', wa_konp_inf-kschl,
'Cond. Rec No:', wa_konp_info-knumh, 'Valid From:', wa_konp_inf, 'Valid To:',
wa_konp_info-datbi, 'Amount:', wa_konp_info-kbetr,   'Unit:',
wa_konp_info-konwa.

  ENDLOOP.

ENDLOOP.
```

Notes

Further information

Functionality

Relationships

Example

Notes

Further information

Attribute

Private attribute

Attrib.	Cat	Description	Ref. Type	Init. value
INDEX	Inst		TYPE I	
GT_MAPPING	Inst		TYPE MAPPING_TABLE	
LT_MAPPING	Inst		TYPE MAPPING_TABLE	
WA_MAPPING	Inst		INST	
GT_PRICPRO_CTYP	Inst		INST	
WA_PRICPRO_CTYP	Inst		INST	
GT_T683V	Inst		INST	
WA_T683V	Inst		INST	
GT_T685	Inst		INST	
WA_T685	Inst		INST	
S_VKORG	Inst		INST	
WA_VKORG	Inst		INST	
S_VTWEG	Inst		INST	
WA_VTWEG	Inst		INST	
S_SPART	Inst		INST	
WA_SPART	Inst		INST	
LINE_COUNT	Inst		TYPE I	
WITH_TAB_ATABLE	Inst		INST	
WITH_TAB_I_KONP	Inst		INST	
WITH_LIN_ATABLE	Inst		INST	
WITH_LIN_I_KONP	Inst		INST	
WITH_CLAUSE_ATABLE	Inst		TYPE STRING	
WITH_CLAUSE_I_KONP	Inst		TYPE STRING	
LV_WORK_AREA_COMPONENT	Inst		TYPE STRING	
GT_KONP_PTR	Inst		TYPE REF TO DATA	
GT_ATABLE_PTR	Inst		TYPE REF TO DATA	
WA_KONP_PTR	Inst		TYPE REF TO DATA	
WA_ATABLE_PTR	Inst		TYPE REF TO DATA	
KEY_TABLE	Inst		INST	
WA_KEY_TABLE	Inst		INST	
P_RANGE_TYPE	Inst		TYPE I	
P_TRAVERSAL	Inst		TYPE GT_TRAVERSAL	
WA_TRAVERSAL	Inst		INST	
TABLE_SIZE	Inst		TYPE P	
CREATE_ATABLE	Inst	A_TABLE Object	TYPE REF TO ZCL_ATABLE_OBJECT	
GT_T682I	Inst		INST	
WA_T682I	Inst		INST	
A_TABLE_LIST	Inst	ATABLE PROCESSING	INST	
A_TABLE_ELEM	Inst		INST	
LIN	Inst		TYPE P	
GT_WITH_LEFT	Inst		INST	
WA_WITH_LEFT	Inst		INST	

Intern. types

Private types

Cat	Cat	Ref. Type	Description
WHERE_TYP	Type	EDPLINE	
TY_LIST_TYPE	Type		
TY_PRICPRO_CTYP	Type		
TY_T683V	Type		
TY_T685	Type		

Cat	Cat	Ref. Type	Description
TY_VKORG	Type		
TY_VTWEG	Type		
TY_SPART	Type		
LEFT_SIDE	Type		

Methods

Public methods

CONSTRUCTOR

Description: Creates A_TABLES using ZCL_ATABLE_OBJECT
Instance mthd

Importing parameter

```
A_TABLE_MAIN TYPE ANY TABLE
GT_MAPPING TYPE MAPPING_TABLE
RANGE_TYPE TYPE I DEFAULT 0 OPTIONAL
P_VKORG TYPE VKORG  OPTIONAL (Sales Organization)
P_VTWEG TYPE VTWEG  OPTIONAL (Distribution Channel)
P_SPART TYPE SPART  OPTIONAL (Division)
P_PARKEY TYPE GT_PARKEY  OPTIONAL (ATABLES using Partial Keys)
```

```
* Populate the A_TABLES                                              *
METHOD constructor.
*  Copy to a local table since you cannot change the GT_MAPPING      *
* parameter.   It is a constructor import parameter.                  *
  lt_mapping[] = gt_mapping[].
* Get the optional parameter information for Sales Organization      *
  CLEAR wa_vkorg.
  wa_vkorg-sign = 'I'.
  wa_vkorg-option = 'EQ'.
  wa_vkorg-low = p_vkorg.
  APPEND wa_vkorg TO s_vkorg.
* Get the optional parameter information for Distribution Channel    *
  CLEAR wa_vtweg.
  wa_vtweg-sign = 'I'.
  wa_vtweg-option = 'EQ'.
  wa_vtweg-low = p_vtweg.
  APPEND wa_vtweg TO s_vtweg.
* Get the optional parameter information for Division                *
  CLEAR wa_spart.
  wa_spart-sign = 'I'.
  wa_spart-option = 'EQ'.
  wa_spart-low = p_spart.
  APPEND wa_spart TO s_spart.
* Select Pricing Procedure there is no partial key parameter         *
  IF LINES( p_parkey ) = 0.
    SELECT * FROM t683v
    INTO CORRESPONDING FIELDS
    OF TABLE gt_t683v
     WHERE vkorg IN s_vkorg
     AND vtweg IN s_vtweg
     AND spart IN s_spart.
    IF sy-subrc = 0 AND LINES( gt_t683v ) > 0 .
```

```
      READ TABLE gt_t683v INTO wa_t683v INDEX 1.
      IF sy-subrc = 0.
*  Select Condition Types for the Pricing Procedure                    *
        SELECT * FROM ztc_pricpro_ctyp
        INTO CORRESPONDING FIELDS
          OF TABLE gt_pricpro_ctyp
          FOR ALL ENTRIES IN gt_t683v
          WHERE kalsm = gt_t683v-kalsm.
        IF sy-subrc = 0 AND LINES( gt_pricpro_ctyp ) > 0.
*  Select Access Sequences for the Condition Types                     *
          #SELECT * FROM t685
            #INTO CORRESPONDING FIELDS
                   OF TABLE gt_t685
                   FOR ALL ENTRIES IN gt_pricpro_ctyp
          #WHERE kschl = gt_pricpro_ctyp-kschl.
          IF sy-subrc = 0 AND LINES( gt_t685 ) > 0.
            REFRESH lt_mapping.
            LOOP AT gt_t685 INTO wa_t685.
              CLEAR wa_mapping.
              wa_mapping-acc_seq = wa_t685-kozgf.
              wa_mapping-map_ord = wa_t685-kschl.
              APPEND wa_mapping TO lt_mapping.
            ENDLOOP.
          ENDIF.
        ENDIF.
      ENDIF.
    ENDIF.
  ENDIF.
*   Get the access sequence from the mapping table                     *
  SELECT * FROM t682i INTO CORRESPONDING FIELDS OF TABLE gt_t682i
  FOR ALL ENTRIES IN lt_mapping
     WHERE kvewe = 'A'
     AND kappl = 'V'
     AND kozgf = lt_mapping-acc_seq.
  IF sy-subrc = 0.
* Read the access sequence table and get the A_TABLE to create, also  *
* passing the mapping element, since ZM07 will be mapped to ZMSD and  *
* ZMVD respectively, since these are the conditions that the A_TABLE  *
* contain for discounts. Typically, the mapping is one-to-one except  *
* in the case of discounts.                                           *
    DATA: wa_mapping LIKE LINE OF lt_mapping.
    LOOP AT lt_mapping INTO wa_mapping.
      LOOP AT gt_t682i INTO wa_t682i
         WHERE kozgf = wa_mapping-acc_seq.
*   Create an A_TABLE object.                                          *
        CREATE OBJECT create_atable
          EXPORTING
            p_kvewe      = wa_t682i-kvewe
            p_kappl      = wa_t682i-kappl
            p_kozgf      = wa_t682i-kozgf
            p_kolnr      = wa_t682i-kolnr
            p_kotabnr    = wa_t682i-kotabnr
            p_main_table = a_table_main
            p_range_type = range_type
```

```
            p_mapto      = wa_mapping-map_ord
            p_parkey     = p_parkey.
*   Get the size of the A_TABLE                                          *
        CALL METHOD create_atable->get_size
          IMPORTING
            p_size = table_size.
*   Add ATABLE object to a LIST                                          *
        IF table_size > 0.
          a_table_elem-ztable_pointer = create_atable.
          a_table_elem-ztable_cond = wa_mapping-map_ord.
          a_table_elem-ztable_size = table_size.
          CLEAR a_table_elem-ztable_name.
          CONCATENATE 'a' wa_t682i-kotabnr INTO a_table_elem-ztable_name.
          APPEND a_table_elem TO a_table_list.
        ENDIF.
      ENDLOOP.
    ENDLOOP.
  ENDIF.
```

GET_KONP_INFO

Description: Gets the pricing information according to mapping order
Instance mthd

Importing parameter

```
WORKAREA TYPE ANY
GT_MAPPING TYPE MAPPING_TABLE
```

Exporting parameter

```
GT_KONP_INFO TYPE KONP_TABLE (Return basic price information)
GT_ATABLE_PTR TYPE REF TO DATA (Return ATABLE table if needed)
GT_KONP_PTR TYPE REF TO DATA (Return KONP table if needed)
```

```
  METHOD get_konp_info.
    FIELD-SYMBOLS: <atable_lt_outtab> TYPE ANY TABLE,
                   <atable_ls_outtab> TYPE ANY,
                   <i_konp_lt_outtab> TYPE ANY TABLE,
                   <i_konp_ls_outtab> TYPE ANY,
                   <a_table_deep> TYPE ty_list_type.
    FIELD-SYMBOLS: <non_existent_field_check> TYPE ANY,
                   <main_field> TYPE ANY,
                   <main_workarea> TYPE ANY.
    DATA: wa_konp_info LIKE LINE OF gt_konp_info.
    REFRESH gt_konp_info.
*  Using Mapping table for the order in which to read the ATABLES     *
    LOOP AT gt_mapping INTO wa_mapping.
*  Read the ATABLE_LIST as a Deep Structure based on the mapping table*
      LOOP AT a_table_list ASSIGNING <a_table_deep>
        WHERE ztable_cond = wa_mapping-map_ord.
        CALL METHOD <a_table_deep>-ztable_pointer->get_keys
          IMPORTING
            table_keys = key_table.
        CALL METHOD <a_table_deep>-ztable_pointer->get_tables
          IMPORTING
            p_gt_konp_ptr   = gt_konp_ptr
            p_gt_atable_ptr = gt_atable_ptr
```

```
            p_wa_konp_ptr   = wa_konp_ptr
            p_wa_atable_ptr = wa_atable_ptr.
*  Clear the With Table                                                 *
       REFRESH with_tab_atable.
*  Add the application selection to the with key clause                 *
       CLEAR with_lin_atable.
       CONCATENATE 'kappl' '= ''V ''' INTO with_lin_atable SEPARATED BY space.
       APPEND with_lin_atable TO with_tab_atable.
*  Add the access sequence to the with key clause                       *
       CLEAR with_lin_atable.
       CONCATENATE ' kschl =' 'WA_MAPPING-MAP_ORD' INTO with_lin_atable SEPARATED BY
       APPEND with_lin_atable TO with_tab_atable.
       REFRESH gt_with_left.
*  Loop through each key field and add to the with key clause of the  *
*  read table statement.                                               *
       LOOP AT key_table INTO wa_key_table.
*  =========  I    M    P    O    R    T    A    N     T  =========   *
*  Skip the key field component if it does not exist in the main      *
*  internal table. UNASSIGN is used to reinitialize field symbol for  *
*  the next pass. The check for the sy-subrc must come right after    *
*  the last ASSIGN statement                                           *
*         ASSIGN workarea TO <main_workarea>.
*         ASSIGN wa_key_table-qufna TO <main_field>.
*         ASSIGN COMPONENT <main_field>
*         OF STRUCTURE <main_workarea>
*         TO <non_existent_field_check>.
*         IF sy-subrc <> 0.
*           UNASSIGN <main_workarea>.
*           UNASSIGN <non_existent_field_check>.
**           CONTINUE.
*         ELSE.
*           UNASSIGN <main_workarea>.
*           UNASSIGN <non_existent_field_check>.
*         ENDIF.
*  Clear the work area for the left side                               *
        CLEAR wa_with_left.
*  Add the condition field component to the main table. For example,  *
*  GT_MAIN_TABLE-VKORG. Used in the next statement                     *
        CLEAR lv_work_area_component.
        CONCATENATE
          'WORKAREA'
          '-'
          wa_key_table-qufna
        INTO lv_work_area_component.
*  Add condition field conditions to the with key clause. For example,*
*  VKORG = GT_MAIN_TABLE-VKORG.                                        *
        CONCATENATE
          ' '
          wa_key_table-qufna
          '='
          lv_work_area_component
        INTO with_lin_atable SEPARATED BY space.
        APPEND with_lin_atable TO with_tab_atable.
        CLEAR wa_with_left.
```

```
          MOVE wa_key_table-qufna TO wa_with_left-left_var.
          APPEND wa_with_left TO gt_with_left.
        ENDLOOP.
* Clear the with key Clause                                          *
        CLEAR with_clause_atable.
        CLEAR with_clause_i_konp.
*  Convert the with key Table to a With key clause                    *
        CONCATENATE LINES OF with_tab_atable INTO with_clause_atable.
        CONCATENATE 'knumh ='
                    '<atable_lt_outtab>-knumh' INTO with_clause_i_konp
                    SEPARATED BY space.
        ASSIGN gt_atable_ptr->* TO <atable_lt_outtab>.
        ASSIGN wa_atable_ptr->* TO <atable_ls_outtab>.
        ASSIGN gt_konp_ptr->* TO <i_konp_lt_outtab>.
        ASSIGN wa_konp_ptr->* TO <i_konp_ls_outtab>.
        FIELD-SYMBOLS: <fs1> TYPE ANY,
                       <fs2> TYPE ANY,
                       <fs3> TYPE ANY,
                       <fs4> TYPE ANY,
                       <fs5> TYPE ANY,
                       <fs6> TYPE ANY,
                       <fs7> TYPE ANY,
                       <fs8> TYPE ANY,
                       <fs9> TYPE ANY,
                       <fs10> TYPE ANY,
                       <fs11> TYPE ANY,
                       <fs12> TYPE ANY,
                       <fs13> TYPE ANY,
                       <fs14> TYPE ANY,
                       <fsvar1> TYPE ANY,
                       <fsvar2> TYPE ANY,
                       <atable_ls_datbi> TYPE ANY,
                       <atable_ls_datab> TYPE ANY,
                       <atable_ls_knumh> TYPE ANY,
                       <konp_ls_kbetr> TYPE ANY,
                       <konp_ls_konwa> TYPE ANY,
                       <konp_ls_kpein> TYPE ANY,
                       <konp_ls_kmein> TYPE ANY.
        DATA: var1 TYPE string,
              var2 TYPE string,
              var3 TYPE string,
              var4 TYPE string,
              var5 TYPE string,
              var6 TYPE string,
              var7 TYPE string,
              var8 TYPE string,
              var9 TYPE string,
              var10 TYPE string,
              var11 TYPE string,
              var12 TYPE string,
              var13 TYPE string,
              var14 TYPE string,
              knumh TYPE string.
        knumh = 'KNUMH'.
```

```
        DESCRIBE TABLE gt_with_left LINES line_count.
*  Supports up to 13 keys on a READ TABLE                                *
        IF line_count >= 1 AND line_count <= 13.
*  1 Key(s) on an ATABLE                                                 *
          IF line_count = 1.
            LOOP AT gt_with_left INTO wa_with_left.
              IF sy-tabix = 1.
                var1 = wa_with_left-left_var.
              ENDIF.
            ENDLOOP.
            var2 = 'KSCHL'.
            ASSIGN COMPONENT var1 OF STRUCTURE workarea TO <fs1>.
            ASSIGN <a_table_deep>-ztable_cond TO <fs2>.
            READ TABLE <atable_lt_outtab>
              WITH KEY
                 (var1) = <fs1>
                 (var2) = <fs2>
                  ASSIGNING <atable_ls_outtab>.
          ENDIF.
*  2 Key(s) on an ATABLE                                                 *
          IF line_count = 2.
            LOOP AT gt_with_left INTO wa_with_left.
              IF sy-tabix = 1.
                var1 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 2.
                var2 = wa_with_left-left_var.
              ENDIF.
            ENDLOOP.
            var3 = 'KSCHL'.
            ASSIGN COMPONENT var1 OF STRUCTURE workarea TO <fs1>.
            ASSIGN COMPONENT var2 OF STRUCTURE workarea TO <fs2>.
            ASSIGN <a_table_deep>-ztable_cond TO <fs3>.
            READ TABLE <atable_lt_outtab>
              WITH KEY
                 (var1) = <fs1>
                 (var2) = <fs2>
                 (var3) = <fs3>
                  ASSIGNING <atable_ls_outtab>.
          ENDIF.
*  3 Key(s) on an ATABLE                                                 *
          IF line_count = 3.
            LOOP AT gt_with_left INTO wa_with_left.
              IF sy-tabix = 1.
                var1 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 2.
                var2 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 3.
                var3 = wa_with_left-left_var.
              ENDIF.
            ENDLOOP.
            var4 = 'KSCHL'.
```

```
            ASSIGN COMPONENT var1 OF STRUCTURE workarea TO <fs1>.
            ASSIGN COMPONENT var2 OF STRUCTURE workarea TO <fs2>.
            ASSIGN COMPONENT var3 OF STRUCTURE workarea TO <fs3>.
            ASSIGN <a_table_deep>-ztable_cond TO <fs4>.
            READ TABLE <atable_lt_outtab>
              WITH KEY
                 (var1) = <fs1>
                 (var2) = <fs2>
                 (var3) = <fs3>
                 (var4) = <fs4>
                  ASSIGNING <atable_ls_outtab>.
          ENDIF.
*  4 Key(s) on an ATABLE                                              *
          IF line_count = 4.
            LOOP AT gt_with_left INTO wa_with_left.
              IF sy-tabix = 1.
                var1 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 2.
                var2 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 3.
                var3 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 4.
                var4 = wa_with_left-left_var.
              ENDIF.
            ENDLOOP.
            var5 = 'KSCHL'.
            ASSIGN COMPONENT var1 OF STRUCTURE workarea TO <fs1>.
            ASSIGN COMPONENT var2 OF STRUCTURE workarea TO <fs2>.
            ASSIGN COMPONENT var3 OF STRUCTURE workarea TO <fs3>.
            ASSIGN COMPONENT var4 OF STRUCTURE workarea TO <fs4>.
            ASSIGN <a_table_deep>-ztable_cond TO <fs5>.
            READ TABLE <atable_lt_outtab>
              WITH KEY
                 (var1) = <fs1>
                 (var2) = <fs2>
                 (var3) = <fs3>
                 (var4) = <fs4>
                 (var5) = <fs5>
                  ASSIGNING <atable_ls_outtab>.
          ENDIF.
*  5 Key(s) on an ATABLE                                              *
          IF line_count = 5.
            LOOP AT gt_with_left INTO wa_with_left.
              IF sy-tabix = 1.
                var1 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 2.
                var2 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 3.
                var3 = wa_with_left-left_var.
```

```
              ENDIF.
              IF sy-tabix = 4.
                var4 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 5.
                var5 = wa_with_left-left_var.
              ENDIF.
            ENDLOOP.
            var6 = 'KSCHL'.
            ASSIGN COMPONENT var1 OF STRUCTURE workarea TO <fs1>.
            ASSIGN COMPONENT var2 OF STRUCTURE workarea TO <fs2>.
            ASSIGN COMPONENT var3 OF STRUCTURE workarea TO <fs3>.
            ASSIGN COMPONENT var4 OF STRUCTURE workarea TO <fs4>.
            ASSIGN COMPONENT var5 OF STRUCTURE workarea TO <fs5>.
            ASSIGN <a_table_deep>-ztable_cond TO <fs6>.
            READ TABLE <atable_lt_outtab>
              WITH KEY
                 (var1) = <fs1>
                 (var2) = <fs2>
                 (var3) = <fs3>
                 (var4) = <fs4>
                 (var5) = <fs5>
                 (var6) = <fs6>
                  ASSIGNING <atable_ls_outtab>.
          ENDIF.
*  6 Key(s) on an ATABLE                                                *
          IF line_count = 6.
            LOOP AT gt_with_left INTO wa_with_left.
              IF sy-tabix = 1.
                var1 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 2.
                var2 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 3.
                var3 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 4.
                var4 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 5.
                var5 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 6.
                var6 = wa_with_left-left_var.
              ENDIF.
            ENDLOOP.
            var7 = 'KSCHL'.
            ASSIGN COMPONENT var1 OF STRUCTURE workarea TO <fs1>.
            ASSIGN COMPONENT var2 OF STRUCTURE workarea TO <fs2>.
            ASSIGN COMPONENT var3 OF STRUCTURE workarea TO <fs3>.
            ASSIGN COMPONENT var4 OF STRUCTURE workarea TO <fs4>.
            ASSIGN COMPONENT var5 OF STRUCTURE workarea TO <fs5>.
            ASSIGN COMPONENT var6 OF STRUCTURE workarea TO <fs6>.
```

```
            ASSIGN <a_table_deep>-ztable_cond TO <fs7>.
            READ TABLE <atable_lt_outtab>
              WITH KEY
                 (var1) = <fs1>
                 (var2) = <fs2>
                 (var3) = <fs3>
                 (var4) = <fs4>
                 (var5) = <fs5>
                 (var6) = <fs6>
                 (var7) = <fs7>
                  ASSIGNING <atable_ls_outtab>.
          ENDIF.
*  7 Key(s) on an ATABLE                                                *
          IF line_count = 7.
            LOOP AT gt_with_left INTO wa_with_left.
              IF sy-tabix = 1.
                var1 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 2.
                var2 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 3.
                var3 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 4.
                var4 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 5.
                var5 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 6.
                var6 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 7.
                var7 = wa_with_left-left_var.
              ENDIF.
            ENDLOOP.
            var8 = 'KSCHL'.
            ASSIGN COMPONENT var1 OF STRUCTURE workarea TO <fs1>.
            ASSIGN COMPONENT var2 OF STRUCTURE workarea TO <fs2>.
            ASSIGN COMPONENT var3 OF STRUCTURE workarea TO <fs3>.
            ASSIGN COMPONENT var4 OF STRUCTURE workarea TO <fs4>.
            ASSIGN COMPONENT var5 OF STRUCTURE workarea TO <fs5>.
            ASSIGN COMPONENT var6 OF STRUCTURE workarea TO <fs6>.
            ASSIGN COMPONENT var7 OF STRUCTURE workarea TO <fs7>.
            ASSIGN <a_table_deep>-ztable_cond TO <fs8>.
            READ TABLE <atable_lt_outtab>
              WITH KEY
                 (var1) = <fs1>
                 (var2) = <fs2>
                 (var3) = <fs3>
                 (var4) = <fs4>
                 (var5) = <fs5>
                 (var6) = <fs6>
```

```
                (var7) = <fs7>
                (var8) = <fs8>
                 ASSIGNING <atable_ls_outtab>.
          ENDIF.
*  8 Key(s) on an ATABLE                                                *
          IF line_count = 8.
            LOOP AT gt_with_left INTO wa_with_left.
              IF sy-tabix = 1.
                var1 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 2.
                var2 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 3.
                var3 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 4.
                var4 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 5.
                var5 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 6.
                var6 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 7.
                var7 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 8.
                var8 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 9.
                var9 = wa_with_left-left_var.
              ENDIF.
            ENDLOOP.
            var9 = 'KSCHL'.
            ASSIGN COMPONENT var1 OF STRUCTURE workarea TO <fs1>.
            ASSIGN COMPONENT var2 OF STRUCTURE workarea TO <fs2>.
            ASSIGN COMPONENT var3 OF STRUCTURE workarea TO <fs3>.
            ASSIGN COMPONENT var4 OF STRUCTURE workarea TO <fs4>.
            ASSIGN COMPONENT var5 OF STRUCTURE workarea TO <fs5>.
            ASSIGN COMPONENT var6 OF STRUCTURE workarea TO <fs6>.
            ASSIGN COMPONENT var7 OF STRUCTURE workarea TO <fs7>.
            ASSIGN COMPONENT var8 OF STRUCTURE workarea TO <fs8>.
            ASSIGN <a_table_deep>-ztable_cond TO <fs9>.
            READ TABLE <atable_lt_outtab>
              WITH KEY
                 (var1) = <fs1>
                 (var2) = <fs2>
                 (var3) = <fs3>
                 (var4) = <fs4>
                 (var5) = <fs5>
                 (var6) = <fs6>
                 (var7) = <fs7>
```

```
                (var8) = <fs8>
                (var9) = <fs9>
                 ASSIGNING <atable_ls_outtab>.
          ENDIF.
*  9 Key(s) on an ATABLE                                                *
          IF line_count = 9.
            LOOP AT gt_with_left INTO wa_with_left.
              IF sy-tabix = 1.
                var1 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 2.
                var2 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 3.
                var3 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 4.
                var4 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 5.
                var5 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 6.
                var6 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 7.
                var7 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 8.
                var8 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 9.
                var9 = wa_with_left-left_var.
              ENDIF.
            ENDLOOP.
            var10 = 'KSCHL'.
            ASSIGN COMPONENT var1 OF STRUCTURE workarea TO <fs1>.
            ASSIGN COMPONENT var2 OF STRUCTURE workarea TO <fs2>.
            ASSIGN COMPONENT var3 OF STRUCTURE workarea TO <fs3>.
            ASSIGN COMPONENT var4 OF STRUCTURE workarea TO <fs4>.
            ASSIGN COMPONENT var5 OF STRUCTURE workarea TO <fs5>.
            ASSIGN COMPONENT var6 OF STRUCTURE workarea TO <fs6>.
            ASSIGN COMPONENT var7 OF STRUCTURE workarea TO <fs7>.
            ASSIGN COMPONENT var8 OF STRUCTURE workarea TO <fs8>.
            ASSIGN COMPONENT var9 OF STRUCTURE workarea TO <fs9>.
            ASSIGN <a_table_deep>-ztable_cond TO <fs10>.
            READ TABLE <atable_lt_outtab>
              WITH KEY
                 (var1) = <fs1>
                 (var2) = <fs2>
                 (var3) = <fs3>
                 (var4) = <fs4>
                 (var5) = <fs5>
                 (var5) = <fs5>
```

```
                (var6) = <fs6>
                (var7) = <fs7>
                (var8) = <fs8>
                (var9) = <fs9>
                (var10) = <fs10>
                 ASSIGNING <atable_ls_outtab>.
          ENDIF.
*  10 Key(s) on an ATABLE                                                *
          IF line_count = 10.
            LOOP AT gt_with_left INTO wa_with_left.
              IF sy-tabix = 1.
                var1 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 2.
                var2 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 3.
                var3 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 4.
                var4 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 5.
                var5 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 6.
                var6 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 7.
                var7 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 8.
                var8 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 9.
                var9 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 10.
                var10 = wa_with_left-left_var.
              ENDIF.
            ENDLOOP.
            var11 = 'KSCHL'.
            ASSIGN COMPONENT var1 OF STRUCTURE workarea TO <fs1>.
            ASSIGN COMPONENT var2 OF STRUCTURE workarea TO <fs2>.
            ASSIGN COMPONENT var3 OF STRUCTURE workarea TO <fs3>.
            ASSIGN COMPONENT var4 OF STRUCTURE workarea TO <fs4>.
            ASSIGN COMPONENT var5 OF STRUCTURE workarea TO <fs5>.
            ASSIGN COMPONENT var6 OF STRUCTURE workarea TO <fs6>.
            ASSIGN COMPONENT var7 OF STRUCTURE workarea TO <fs7>.
            ASSIGN COMPONENT var8 OF STRUCTURE workarea TO <fs8>.
            ASSIGN COMPONENT var9 OF STRUCTURE workarea TO <fs9>.
            ASSIGN COMPONENT var10 OF STRUCTURE workarea TO <fs10>.
            ASSIGN <a_table_deep>-ztable_cond TO <fs11>.
            READ TABLE <atable_lt_outtab>
```

```
              WITH KEY
                 (var1) = <fs1>
                 (var2) = <fs2>
                 (var3) = <fs3>
                 (var4) = <fs4>
                 (var5) = <fs5>
                 (var5) = <fs5>
                 (var6) = <fs6>
                 (var7) = <fs7>
                 (var8) = <fs8>
                 (var9) = <fs9>
                 (var10) = <fs10>
                 (var11) = <fs11>
                  ASSIGNING <atable_ls_outtab>.
          ENDIF.
*  11 Key(s) on an ATABLE                                                *
          IF line_count = 11.
            LOOP AT gt_with_left INTO wa_with_left.
              IF sy-tabix = 1.
                var1 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 2.
                var2 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 3.
                var3 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 4.
                var4 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 5.
                var5 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 6.
                var6 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 7.
                var7 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 8.
                var8 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 9.
                var9 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 10.
                var10 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 11.
                var11 = wa_with_left-left_var.
              ENDIF.
            ENDLOOP.
            var12 = 'KSCHL'.
            ASSIGN COMPONENT var1 OF STRUCTURE workarea TO <fs1>.
```

```
            ASSIGN COMPONENT var2 OF STRUCTURE workarea TO <fs2>.
            ASSIGN COMPONENT var3 OF STRUCTURE workarea TO <fs3>.
            ASSIGN COMPONENT var4 OF STRUCTURE workarea TO <fs4>.
            ASSIGN COMPONENT var5 OF STRUCTURE workarea TO <fs5>.
            ASSIGN COMPONENT var6 OF STRUCTURE workarea TO <fs6>.
            ASSIGN COMPONENT var7 OF STRUCTURE workarea TO <fs7>.
            ASSIGN COMPONENT var8 OF STRUCTURE workarea TO <fs8>.
            ASSIGN COMPONENT var9 OF STRUCTURE workarea TO <fs9>.
            ASSIGN COMPONENT var10 OF STRUCTURE workarea TO <fs10>.
            ASSIGN COMPONENT var11 OF STRUCTURE workarea TO <fs11>.
            ASSIGN <a_table_deep>-ztable_cond TO <fs12>.
            READ TABLE <atable_lt_outtab>
              WITH KEY
                 (var1) = <fs1>
                 (var2) = <fs2>
                 (var3) = <fs3>
                 (var4) = <fs4>
                 (var5) = <fs5>
                 (var5) = <fs5>
                 (var6) = <fs6>
                 (var7) = <fs7>
                 (var8) = <fs8>
                 (var9) = <fs9>
                 (var10) = <fs10>
                 (var11) = <fs11>
                 (var12) = <fs12>
                  ASSIGNING <atable_ls_outtab>.
          ENDIF.
*  12 Key(s) on an ATABLE                                                *
          IF line_count = 12.
            LOOP AT gt_with_left INTO wa_with_left.
              IF sy-tabix = 1.
                var1 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 2.
                var2 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 3.
                var3 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 4.
                var4 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 5.
                var5 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 6.
                var6 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 7.
                var7 = wa_with_left-left_var.
              ENDIF.
              IF sy-tabix = 8.
                var8 = wa_with_left-left_var.
```

```
            ENDIF.
            IF sy-tabix = 9.
              var9 = wa_with_left-left_var.
            ENDIF.
            IF sy-tabix = 10.
              var10 = wa_with_left-left_var.
            ENDIF.
            IF sy-tabix = 11.
              var11 = wa_with_left-left_var.
            ENDIF.
            IF sy-tabix = 12.
              var12 = wa_with_left-left_var.
            ENDIF.
          ENDLOOP.
          var13 = 'KSCHL'.
          ASSIGN COMPONENT var1 OF STRUCTURE workarea TO <fs1>.
          ASSIGN COMPONENT var2 OF STRUCTURE workarea TO <fs2>.
          ASSIGN COMPONENT var3 OF STRUCTURE workarea TO <fs3>.
          ASSIGN COMPONENT var4 OF STRUCTURE workarea TO <fs4>.
          ASSIGN COMPONENT var5 OF STRUCTURE workarea TO <fs5>.
          ASSIGN COMPONENT var6 OF STRUCTURE workarea TO <fs6>.
          ASSIGN COMPONENT var7 OF STRUCTURE workarea TO <fs7>.
          ASSIGN COMPONENT var8 OF STRUCTURE workarea TO <fs8>.
          ASSIGN COMPONENT var9 OF STRUCTURE workarea TO <fs9>.
          ASSIGN COMPONENT var10 OF STRUCTURE workarea TO <fs10>.
          ASSIGN COMPONENT var11 OF STRUCTURE workarea TO <fs11>.
          ASSIGN COMPONENT var12 OF STRUCTURE workarea TO <fs12>.
          ASSIGN <a_table_deep>-ztable_cond TO <fs13>.
          READ TABLE <atable_lt_outtab>
            WITH KEY
               (var1) = <fs1>
               (var2) = <fs2>
               (var3) = <fs3>
               (var4) = <fs4>
               (var5) = <fs5>
               (var5) = <fs5>
               (var6) = <fs6>
               (var7) = <fs7>
               (var8) = <fs8>
               (var9) = <fs9>
               (var10) = <fs10>
               (var11) = <fs11>
               (var12) = <fs12>
               (var13) = <fs13>
                ASSIGNING <atable_ls_outtab>.
        ENDIF.
*  13 Key(s) on an ATABLE                                              *
        IF line_count = 13.
          LOOP AT gt_with_left INTO wa_with_left.
            IF sy-tabix = 1.
              var1 = wa_with_left-left_var.
            ENDIF.
            IF sy-tabix = 2.
              var2 = wa_with_left-left_var.
```

```
  ENDIF.
  IF sy-tabix = 3.
    var3 = wa_with_left-left_var.
  ENDIF.
  IF sy-tabix = 4.
    var4 = wa_with_left-left_var.
  ENDIF.
  IF sy-tabix = 5.
    var5 = wa_with_left-left_var.
  ENDIF.
  IF sy-tabix = 6.
    var6 = wa_with_left-left_var.
  ENDIF.
  IF sy-tabix = 7.
    var7 = wa_with_left-left_var.
  ENDIF.
  IF sy-tabix = 8.
    var8 = wa_with_left-left_var.
  ENDIF.
  IF sy-tabix = 9.
    var9 = wa_with_left-left_var.
  ENDIF.
  IF sy-tabix = 10.
    var10 = wa_with_left-left_var.
  ENDIF.
  IF sy-tabix = 11.
    var11 = wa_with_left-left_var.
  ENDIF.
  IF sy-tabix = 12.
    var12 = wa_with_left-left_var.
  ENDIF.
ENDLOOP.
var13 = 'KSCHL'.
ASSIGN COMPONENT var1 OF STRUCTURE workarea TO <fs1>.
ASSIGN COMPONENT var2 OF STRUCTURE workarea TO <fs2>.
ASSIGN COMPONENT var3 OF STRUCTURE workarea TO <fs3>.
ASSIGN COMPONENT var4 OF STRUCTURE workarea TO <fs4>.
ASSIGN COMPONENT var5 OF STRUCTURE workarea TO <fs5>.
ASSIGN COMPONENT var6 OF STRUCTURE workarea TO <fs6>.
ASSIGN COMPONENT var7 OF STRUCTURE workarea TO <fs7>.
ASSIGN COMPONENT var8 OF STRUCTURE workarea TO <fs8>.
ASSIGN COMPONENT var9 OF STRUCTURE workarea TO <fs9>.
ASSIGN COMPONENT var10 OF STRUCTURE workarea TO <fs10>.
ASSIGN COMPONENT var11 OF STRUCTURE workarea TO <fs11>.
ASSIGN COMPONENT var12 OF STRUCTURE workarea TO <fs12>.
ASSIGN <a_table_deep>-ztable_cond TO <fs13>.
READ TABLE <atable_lt_outtab>
  WITH KEY
     (var1) = <fs1>
     (var2) = <fs2>
     (var3) = <fs3>
     (var4) = <fs4>
     (var5) = <fs5>
     (var5) = <fs5>
```

```
                (var6) = <fs6>
                (var7) = <fs7>
                (var8) = <fs8>
                (var9) = <fs9>
                (var10) = <fs10>
                (var11) = <fs11>
                (var12) = <fs12>
                (var13) = <fs13>
                 ASSIGNING <atable_ls_outtab>.
          ENDIF.
*  ATABLE record is found, now look for the KONP record                *
          IF sy-subrc = 0.
*  KNUM is the last field in the structure of the ATABLE.  Use it to  *
*  find the corresponding condition the KONP table.                    *
            CLEAR index.
            CLEAR sy-subrc.
            WHILE sy-subrc = 0.
              index = index + 1.
              ASSIGN COMPONENT index OF STRUCTURE <atable_ls_outtab> TO
            ENDWHILE.
*  Get the date range from the ATABLE                                  *
            index = index - 2.
            ASSIGN COMPONENT index OF STRUCTURE <atable_ls_outtab> TO
            index = index - 1.
            ASSIGN COMPONENT index  OF STRUCTURE <atable_ls_outtab> TO
*  Get the condition record from the KONP table. If it is found then  *
*  exit the ATABLE_LIST loop and find the next condition ex/ ZMSD from*
*  the mapping table.                                                  *
            READ TABLE <i_konp_lt_outtab>
              WITH KEY
                 (knumh) = <atable_ls_knumh>
                  ASSIGNING <i_konp_ls_outtab>.
            IF sy-subrc = 0.
              CLEAR wa_konp_info.
              ASSIGN COMPONENT 14 OF STRUCTURE <i_konp_ls_outtab> TO <konp_ls_kbetr>.
              ASSIGN COMPONENT 15 OF STRUCTURE <i_konp_ls_outtab> TO <konp_ls_konwa>.
              ASSIGN COMPONENT 16 OF STRUCTURE <i_konp_ls_outtab> TO <konp_ls_kpein>.
              ASSIGN COMPONENT 17 OF STRUCTURE <i_konp_ls_outtab> TO <konp_ls_kmein>.
              wa_konp_info-aname =  <a_table_deep>-ztable_name.
              wa_konp_info-kschl =  <a_table_deep>-ztable_cond.
              wa_konp_info-knumh = <atable_ls_knumh>.
              wa_konp_info-datbi = <atable_ls_datbi>.
              wa_konp_info-datab = <atable_ls_datab>.
              wa_konp_info-kbetr = <konp_ls_kbetr>.
              wa_konp_info-konwa = <konp_ls_konwa>.
              wa_konp_info-kpein = <konp_ls_kpein>.
              wa_konp_info-kmein = <konp_ls_kmein>.
              APPEND wa_konp_info TO gt_konp_info.
              EXIT.
            ENDIF.
          ENDIF.
        ELSE.
          EXIT.
        ENDIF.
```

```
      ENDLOOP.
    ENDLOOP.
  ENDMETHOD.
```

GET_CUSTOM_MAPPING

Description: Get the mapping based on the pricing procedure
Instance mthd

Exporting parameter

```
 GT_CUSTOM_MAPPING TYPE MAPPING_TABLE
```

```
method GET_CUSTOM_MAPPING.
   gt_custom_mapping = lt_mapping.
```

Redefined Methods

Local Types

```
*"* use this source file for any type declarations (class
*"* definitions, interfaces or data types) you need for method
*"* implementation or private method's signature
```

Local class definitions

```
*"* local class implementation for public class
*"* use this source file for the implementation part of
*"* local helper classes
```

Macros

```
*"* use this source file for any macro definitions you need
*"* in the implementation part of the class
```

Overview