Class: ZCL_ATABLE_OBJECT

Status: Active

Attributes

Description: ATABLE 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_Tables and the corresponding pricing conditions in the KONP file, based on the table Access Sequences in the table T682I.

- All of the methods are called by the class **ZCL_ATABLE_HELPER**
- The constructor is called by the associated **ZCL_ATABLE_HELPER** constructor, for each access sequence in T682I.

Methods:

CONSTRUCTOR: Creates the required A_TABLES and the corresponding pricing conditions (KONP).

Parameters:

P_KWEWE: Usage

P_KAPPL: Application

P_KOZGF: Access Sequence

P_KOLNR: Access Number

P_KOTABNR: Condition Table

P_MAIN_TABLE: Main internal table passed in from the main program and used in the for all entries of the select statement.

P_RANGE_TYPE: 0-Standard, 1-Effective, 2-Ending.

P_MAP_TO: Access sequence mapping / traversal order

GET_SIZE: Returns the number of pricing record condtions (KONP)

Parameters:

P_SIZE: Internal table size.

GET_TABLES: Returns pointers to the ATABLE, KONP tables and the corre sponding work areas.

P_GT_KONP_PTR: Pointer to the pricing condition internal table.

P_GT_ATABLE_PTR: Pointer to the ATABLE internal table.

P_WA_KONP_PTR: Pointer to the pricing condition work area.

P_WA_ATABLE_PTR: Pointer to the ATABLE work area.

GET_KEYS: Returns the keyed indexes for the A_TABLES.

TABLE_KEYS: Key fields.

Attribute

Private attribute

Attrib. Ref. Type	Cat	Description Init. value
LV_RANGE_TYPE TYPE I	Inst	
ATABLE_POINTER TYPE REF TO CL_ABAP_TABLEDESCR	Inst	Runtime Type Services
LIN TYPE P	Inst	
LV_ATABLE_NAME TYPE STRING	Inst	Condition Type
LV_ATABLE_TEMP	Inst	
LV_ATABLE_NAME_KSCHL TYPE STRING	Inst	
LV_ATABLE_WHERE TYPE STRING	Inst	Selection String
GT_T682I	Inst	Conditions: Access Sequences (Generated Form)
WA_T682I	Inst	
LV_KVEWE TYPE T682I-KVEWE	Inst	Usage of the condition table
LV_KAPPL TYPE T682I-KAPPL	Inst	

Attrib.	**Cat**	**Description**
Ref. Type		**Init. value**
LV_KOZGF	Inst	
TYPE T682I-KOZGF		
LV_KOLNR	Inst	
TYPE T682I-KOLNR		
LV_KOTABNR	Inst	
TYPE T682I-KOTABNR		
LV_MAPTO	Inst	
TYPE T682I-KOZGF		
GT_T682Z	Inst	
WA_T682Z	Inst	
LT_PARKEY	Inst	
WA_PARKEY	Inst	
LV_MAIN_TABLE	Inst	
TYPE STRING		
ITAB_TYPE_ATABLE	Inst	
TYPE REF TO CL_ABAP_TABLEDESCR		
ITAB_TYPE_I_KONP	Inst	
TYPE REF TO CL_ABAP_TABLEDESCR		
STRUCT_TYPE_ATABLE	Inst	
TYPE REF TO CL_ABAP_STRUCTDESCR		
STRUCT_TYPE_I_KONP	Inst	
TYPE REF TO CL_ABAP_STRUCTDESCR		
STRUCT_TYPE_P_MAIN	Inst	
TYPE REF TO CL_ABAP_STRUCTDESCR		
DREF_ATABLE_LT_OUTTAB	Inst	
TYPE REF TO DATA		
DREF_I_KONP_LT_OUTTAB	Inst	
TYPE REF TO DATA		
DREF_ATABLE_LS_OUTTAB	Inst	
TYPE REF TO DATA		
DREF_I_KONP_LS_OUTTAB	Inst	
TYPE REF TO DATA		
COMP_TAB_ATABLE	Inst	
TYPE CL_ABAP_STRUCTDESCR=>COMPONENT_TABLE		
COMP_TAB_I_KONP	Inst	
TYPE CL_ABAP_STRUCTDESCR=>COMPONENT_TABLE		
COMP_TAB_P_MAIN	Inst	
TYPE CL_ABAP_STRUCTDESCR=>COMPONENT_TABLE		
WHERE_TAB_ATABLE	Inst	
WHERE_TAB_I_KONP	Inst	

Attrib. Ref. Type	Cat	Description Init. value
WHERE_LIN_ATABLE INST	Inst	
WHERE_LIN_I_KONP	Inst	
WHERE_CLAUSE_ATABLE TYPE STRING	Inst	
WHERE_CLAUSE_I_KONP TYPE STRING	Inst	
LV_MAIN_TABLE_COMPONENT TYPE STRING	Inst	

Intern. types

Private types

Cat	Cat	Ref. Type	Description
WHERE_TYP	Type	EDPLINE	

Methods

Public methods

CONSTRUCTOR

Description: Creates the A_TABLE and the corresponding KONP table
Instance mthd

Importing parameter

```
  VALUE(P_KVEWE) TYPE T682I-KVEWE (Usage of the condition table)
  VALUE(P_KAPPL) TYPE T682I-KAPPL (Application)
  VALUE(P_KOZGF) TYPE T682I-KOZGF (Access sequence)
  VALUE(P_KOLNR) TYPE T682I-KOLNR (Access sequence - Access number)
  VALUE(P_KOTABNR) TYPE T682I-KOTABNR (Condition table)
  VALUE(P_MAIN_TABLE) TYPE ANY TABLE
  VALUE(P_RANGE_TYPE) TYPE I
  VALUE(P_MAPTO) TYPE T682I-KOZGF (Access sequence mapping)
  P_PARKEY TYPE GT_PARKEY  OPTIONAL (ATABLES with partial keys)
```

```
METHOD constructor.
  FIELD-SYMBOLS: <non_existent_field_check> TYPE ANY,
                 <main_field> TYPE ANY,
                 <main_table> TYPE ANY TABLE,
                 <main_workarea> TYPE ANY.
  lv_kvewe = p_kvewe.
  lv_kappl = p_kappl.
  lv_kozgf = p_kozgf.
  lv_kolnr = p_kolnr.
  lv_kotabnr = p_kotabnr.
  lv_range_type = p_range_type.
  lv_mapto = p_mapto.
  lt_parkey[] = p_parkey[].
* Use field symbol for key-component processing                    *
```

```
*---------------------------------------------------------------------*
  ASSIGN p_main_table TO <main_table>.
*  Get the name of the A_TABLE                                        *
  CONCATENATE 'a' lv_kotabnr INTO lv_atable_name.
* Get the name of the condition table                                 *
  CONCATENATE 'i_a' lv_kotabnr '_' lv_kozgf INTO lv_atable_name_kschl.
  TRANSLATE lv_atable_name_kschl TO LOWER CASE.
* Get the keys for the dynamic select   (key fields)                 *
  SELECT * FROM t682z INTO TABLE gt_t682z
     WHERE kvewe = lv_kvewe
     AND   kappl = lv_kappl
     AND   kozgf = lv_kozgf
     AND   kolnr = lv_kolnr.
  IF sy-subrc = 0.
*  Get the keys by the key sequence number                            *
    SORT gt_t682z BY zaehk.
*  Clear the Where Table                                              *
    REFRESH where_tab_atable.
*  Add the application selection to the where clause                  *
    CLEAR where_lin_atable.
    CONCATENATE 'kappl' '= ''V ''' INTO where_lin_atable SEPARATED BY space.
    APPEND where_lin_atable TO where_tab_atable.
*  Add the access sequence to the where clause                        *
    CLEAR where_lin_atable.
    CONCATENATE ' AND kschl =' 'lv_mapto' INTO where_lin_atable SEPARATED BY space.
    APPEND where_lin_atable TO where_tab_atable.
*  Loop through each key field and add to the where clause of the     *
*  select statement.                                                  *
    LOOP AT gt_t682z INTO wa_t682z.
*  ====== P    A    R    T    I    A    L       K    E    Y  ======== *
*  Check to see if a partial key check was requested for a specific   *
*  ATABLE in LT_PARKEY                                                *
      lv_atable_temp = lv_atable_name.
      READ TABLE lt_parkey
            INTO wa_parkey
            WITH KEY atable = lv_atable_temp.
      IF sy-subrc = 0.
*  =========  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                                          *
        LOOP AT <main_table> ASSIGNING <main_workarea>.
          EXIT.
        ENDLOOP.
*  One pass was made through the loop to obtain the work area struc-  *
*  ture.  This could not be done using READ TABLE and an index since  *
*  the main_table type is ANY TABLE. Index operations are not allowed *
*  type ANY TABLE.                                                    *
        IF sy-subrc = 0.
          ASSIGN wa_t682z-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>.
            DELETE gt_t682z.
            CONTINUE.
          ELSE.
            UNASSIGN <main_workarea>.
            UNASSIGN <non_existent_field_check>.
          ENDIF.
        ENDIF.
      ENDIF.
*  Add the condition field component to the main table. For example,  *
*  GT_MAIN_TABLE-VKORG. Used in the next statement                     *
      CLEAR lv_main_table_component.
      CONCATENATE
        'P_MAIN_TABLE'
        '-'
        wa_t682z-qufna
      INTO lv_main_table_component.
*  Add condition field conditions to the where clause. For example,   *
*  VKORG = GT_MAIN_TABLE-VKORG.                                       *
      CONCATENATE
        ' AND'
        wa_t682z-qufna
        '='
        lv_main_table_component
      INTO where_lin_atable SEPARATED BY space.
      APPEND where_lin_atable TO where_tab_atable.
    ENDLOOP.
*  Add the range to the where clause based on the range type          *
    IF lv_range_type = 0 OR lv_range_type = 1.
      CLEAR where_lin_atable.
      CONCATENATE
      ' AND'
      '( DATAB <= P_MAIN_TABLE-MYDATE AND DATBI >= P_MAIN_TABLE-MYDATE )'
      INTO where_lin_atable SEPARATED BY space.
      APPEND where_lin_atable TO where_tab_atable.
    ELSEIF lv_range_type = 2.
      CLEAR where_lin_atable.
      CONCATENATE
      ' AND'
      'DATBI = P_MAIN_TABLE-MYDATE'
      INTO where_lin_atable SEPARATED BY space.
      APPEND where_lin_atable TO where_tab_atable.
    ENDIF.
*  Clear the Where Clause                                             *
    CLEAR where_clause_atable.
    CLEAR where_clause_i_konp.
*  Convert the Where Table to a Where String clause                   *
    CONCATENATE LINES OF where_tab_atable INTO where_clause_atable.
    CONCATENATE 'knumh ='
                '<atable_lt_outtab>-knumh' INTO where_clause_i_konp
                SEPARATED BY space.
*  Dynamically build the table and the corresponding structure        *
```

```
*---------------------------------------------------------------------*
    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,
                   <main_tab> TYPE ANY.
*  Give the structure a name                                          *
    struct_type_atable ?= cl_abap_typedescr=>describe_by_name( lv_atable_name ).
    struct_type_i_konp ?= cl_abap_typedescr=>describe_by_name( 'konp' ).
*  Get the components of the structrue                                *
    comp_tab_atable = struct_type_atable->get_components( ).
    comp_tab_i_konp = struct_type_i_konp->get_components( ).
*  Dynamically create the structure                                   *
    struct_type_atable = cl_abap_structdescr=>create( comp_tab_atable ).
    struct_type_i_konp = cl_abap_structdescr=>create( comp_tab_i_konp ).
*  Dynamically create the table                                       *
    itab_type_atable = cl_abap_tabledescr=>create( struct_type_atable ).
    itab_type_i_konp = cl_abap_tabledescr=>create( struct_type_i_konp ).
*  Create a pointer to the table object and assign to field symbol    *
    CREATE DATA dref_atable_lt_outtab TYPE HANDLE itab_type_atable.
    ASSIGN dref_atable_lt_outtab->* TO <atable_lt_outtab>.
    CREATE DATA dref_i_konp_lt_outtab TYPE HANDLE itab_type_i_konp.
    ASSIGN dref_i_konp_lt_outtab->* TO <i_konp_lt_outtab>.
*  Create a pointer to the structure object and assign to field symbol*
    CREATE DATA dref_atable_ls_outtab TYPE HANDLE struct_type_atable.
    ASSIGN dref_atable_ls_outtab->* TO <atable_ls_outtab>.
    CREATE DATA dref_i_konp_ls_outtab TYPE HANDLE struct_type_i_konp.
    ASSIGN dref_i_konp_ls_outtab->* TO <i_konp_ls_outtab>.
*  Get the number of lines in the main internal table                 *
    DESCRIBE TABLE p_main_table LINES lin.
*  Make sure the internal table is populated.                         *
    IF lin > 0.
*  Create the dynamic select statement populating A_TABLE             *
      TRY.
          SELECT * FROM (lv_atable_name)
          INTO CORRESPONDING FIELDS OF TABLE <atable_lt_outtab>
          FOR ALL ENTRIES IN p_main_table
          WHERE (where_clause_atable).
*  A_TABLE records are found.                                         *
          IF sy-subrc = 0.
*  Get the number of lines in the secondary internal table            *
            DESCRIBE TABLE <atable_lt_outtab> LINES lin.
*  Make sure the internal table is populated.                         *
            IF lin > 0.
*  Create the dynamic select statement populating KONP TABLE          *
              SELECT * FROM konp
              INTO CORRESPONDING FIELDS OF TABLE <i_konp_lt_outtab>
              FOR ALL ENTRIES IN <atable_lt_outtab>
              WHERE (where_clause_i_konp)
              AND loevm_ko = space.
              IF sy-subrc <> 0.
                REFRESH <i_konp_lt_outtab>.
              ENDIF.
            ENDIF.
```

```
          ENDIF.
*  Field might not be found p_main_table                                *
        CATCH cx_sy_dynamic_osql_semantics.
      ENDTRY.
    ENDIF.
  ENDIF.
```

GET_SIZE

Description: Get the size of the A_TABLE
Instance mthd

Exporting parameter

```
 VALUE(P_SIZE) TYPE P
```

```
METHOD GET_SIZE.
  FIELD-SYMBOLS: <i_konp_lt_outtab> TYPE ANY TABLE.
  ASSIGN dref_i_konp_lt_outtab->* TO <i_konp_lt_outtab>.
  DESCRIBE TABLE  <i_konp_lt_outtab> LINES lin.
  IF sy-subrc = 0.
    p_size = lin.
  ENDIF.
```

GET_TABLES

Description: Get the A_TABLE and the corresponding KONP table
Instance mthd

Exporting parameter

```
 P_GT_KONP_PTR TYPE GT_KONP_PTR
 P_GT_ATABLE_PTR TYPE GT_ATABLE_PTR
 P_WA_KONP_PTR TYPE WA_KONP_PTR
 P_WA_ATABLE_PTR TYPE WA_ATABLE_PTR
```

```
METHOD GET_TABLES.
  P_GT_KONP_PTR = dref_i_konp_lt_outtab.
  P_GT_ATABLE_PTR = dref_atable_lt_outtab.
  P_WA_KONP_PTR = dref_i_konp_ls_outtab.
  P_WA_ATABLE_PTR = dref_atable_ls_outtab.
```

GET_KEYS

Description: Get the keys for the READ_TABLE statement
Instance mthd

Exporting parameter

```
 TABLE_KEYS TYPE TY_682Z
```

```
method GET_KEYS.
    table_keys = gt_t682z.
```

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