Functional Module Based Delta Enabled Generic Datasource

Applies to:

SAP R/3, SAP ECC 6.0 and SAP BI NetWeaver 2004s and above. For more information, visit the EDW homepage

Summary

This document explains the process to create Delta enabled Generic Datasource based on Function Module. Here I explained the steps required to use RSAX_BIW_GET_DATA_SIMPLE (Function Module meant for Full Load) to create Delta enable Extractor. Person with or without prior knowledge of ABAP can use this article to understand the working of Function Module based Generic Extractor. Articles explain everything right from the creation of the dummy transparent table to that of enabling Delta of a Datasource. It also describes auxiliary steps like creation of Table Maintenance and TCode creation for direct data entry. If you are looking for the entire steps involved in the creation of Delta Enabled Generic Datasource based on Function Module, this paper will definitely help you doing that.

Author: Debjit Singha

Company: L & T Infotech.

Created on: 08 July 2011

Author Bio

Debjit Singha is currently working with L & T Infotech. He is a Business Intelligence Solution Consultant. His core expertise includes BSP, BI, BO and Widgets. He extensively worked in SAP Crystal Dashboard Design, BI.

Table of Contents

Introduction

This article provides an insight in to the various elements of Functional Module based Extraction. It also help one to build their own Delta enabled Datasource, transparent Table, Table Maintenance etc.

Here I explained the steps required to use RSAX_BIW_GET_DATA_SIMPLE (Function Module meant for Full Load) to create Delta enable Extractor. Person with or without prior knowledge of ABAP can use this article to understand the working of Function Module based Generic Extractor. Articles explain everything right from the creation of the dummy transparent table to that of enabling Delta of a Datasource. It also describes auxiliary steps like creation of Table Maintenance and TCode creation for direct data entry.

There are many articles on Generic Extractors based on Function Module (RSAX_BIW_GET_DATA_SIMPLE). This article shows, how with some workaround we can use the same Function Module to create Delta enabled Datasource.

Document Content:

Creation of Delta Enabled, Function Module Based Datasource.

Auxiliary Steps: Steps to create Transparent Table (dummy).

Auxiliary Steps: Steps to Generate Table Maintenance and create Tcode for record creation.

Scenario

Delta capability of SAP BW system makes it unique. There are situation where we might need to go for Generic Datasource created on functional module. One can find lots of documents on SDN that gives u details about how to achieve this. Delta can be created on Timestamp, Calendar Day and on Numeric Pointer. Mostly creation date (ERDAT) is considered for delta extraction, but in some scenarios we have an added field, Last Change Date (AEDAT). AEDAT keep track of the changes performed on records. So it become necessary to pull changed records along with newly created records. This task can easily performed by created two generic delta enabled Datasources based on view, but this will increase maintenance activity.

The other way is to create generic Datasource based on function module, which is capable of pulling delta considering both creation as well as changed on date.

We will be using sample Function Module (RSAX_BIW_GET_DATA_SIMPLE) to create Delta enabled FM.

Parameters

These are export parameters that have to be supplied with data to the function module. They have to be supplied with data unless they are flagged as optional. We cannot change these parameter values in the function module. We have another type called changing parameters for them.

Function Module RSAX_BIW_GET_DATA_SIMPLE consists of below mentioned parameters.

1. I_REQUNR Request Number. System Interface table (SRSC_S_IF_SIMPLE explained later) pass this parameter to FM.
2. I_DSOURCE Datasource
3. I_MAXSIZE Packet Size. Maximum number of records per packet.
4. I_INITFLAG Initialization call. This parameter is set to ‘X’ when the function module is called up for the first time, then to “ “.

Out of which except I_REQUNR, all other parameters are marked as optional.

Table

Table parameters are especially used to pass internal tables. They also function like changing parameters. They have to be supplied with values unless they are marked as optional.

I_T_SELECT

It contains the selection criteria defined at the info package level as well as it being used during Delta processing it gets populated with present date as High and one date after the last delta execution will become the Low value.

I_T_FIELDS

It contains all the fields in the structure used as extract structure.

E_T_DATA

Structure is same as that of the extract structure of the Datasource.(In our case we mapped it with ZSTR_DS_FM_TEST)

It holds the extracted record set.

Exceptions

Exceptions are for handling situations where in which normal execution of program becomes difficult. For example if there are not entries in the table TAB1 that meet the selection criteria. We need NOT_FOUND exception for this.

NO_MORE_DATA (Module returns the this once there is no more data to be fetched)

ERROR_PASSED_TO_MESS_HANDLER

Execution Process

Function Modules will be called several times in succession during an extraction process

1. Initialization step: Only the request parameters are transferred to the module here. No data is transferred in this step.
2. First read call: in this step extract structure is returned to the I_T_FIELDS (explained later in Internal Processing). System fetched the Maximum number of rows that each packet can have. It is fetched from parameter I_MAXSIZE which in turn gets its value from system Interface table SRSC_S_IF_SIMPLE (explained later). In this phase the values from I_T_SELECT is transferred to the corresponding Range table.
3. Second read call: First it set the Open Cursor Statement and then fetches data according to I_MAXSIZE (it decides the packet size).
4. System iterates the FM again and again until module returns NO_MORE_DATA exception (carries system defined value 1). FM call containing exception wont returns any data.

Steps for Delta Enabled, Function Module Based Datasource

1. Copy the Function group RSAX from SE80, give New function group as ZRSAX_TEST.
2. Copy function module. Deselect all and then select only RSAX_BIW_GET_DATA_SIMPLE name it as ZRSAX_BIW_GET_DATA_SIMPLE_TEST.
3. Go to the Include folder and double-click on LZRSAX_TESTTOP define the structure and Field symbol and internal table as below.

This is the code used in the above mentioned Include

```
function-pool zrsax_test                 message-id sv.

* Include LRSAXD01 can be directly referenced by application APIs !!
include lrsaxd01.

include rsaucmac.

types: begin of x_ztab_ds_fm_test,

ebeln type ebeln,
erdat type erdat,
aedat type aedat,
loekz type loekz,
ntgew type ntgew,
gewei type gewei,


      end of x_ztab_ds_fm_test.

field-symbols: <fs_x_ztab_ds_fm_test> type x_ztab_ds_fm_test.
data : i_ztab_ds_fm_test type standard table of x_ztab_ds_fm_test.
```

Save and activate it.

Open the function module ZRSAX_BIW_GET_DATA_SIMPLE_TEST

```
* Auxiliary Selection criteria structure
  data: l_s_select type srsc_s_select.
```

srsc_s_select is like RSSELECT, which is a structure interface selection criteria made up of components: FIELDNM, SIGN, OPTION, LOW, HIGH. This is used later in our code to take input from System interface table SRSC_S_IF_SIMPLE and then pass it to range table.

```
* Maximum number of lines for DB table
  statics: s_s_if type SRSC_S_IF_SIMPLE ,
```

s_s_if takes input from system defined Interface table SRSC_S_IF_SIMPLE contain fields like REQUNR, DSOURCE, MAXSIZE, INITFLAG, READONLY, T_SELECT (system range table), T_FIELDS (contains all the fields in the structure used as extract structure). Later it is used to fill the range table (like l_r_ebeln).

```
* cursor
  s_cursor type cursor.
```

Kindly refer Appendix for more details on cursor. s_cursor is used for operations like open cursor , fetch and close cursor. This is the “fetch next cursor” is the commend where we actually fetch data from database.

```
* Select ranges
  ranges: l_r_ebeln  for zstr_ds_fm_test-ebeln,
 l_r_erdat  for zstr_ds_fm_test-erdat,
l_r_zdate for zstr_ds_fm_test-zdate .
```

We define range table to take inputs from the infopackage selection screen and in case of delta it takes the range (range of date last delta execution till this current date) from interface SRSC_S_IF_SIMPLE-T_SELECT.

```
   * Initialization mode (first call by SAPI) or data transfer mode
 * (following calls) ?
  if i_initflag = sbiwa_c_flag_on.
```

sbiwa_c_flag_on is a system flag set at “X”. During initialization phase this if condition will be true as i_initflag also holds”X” (from S_S_IF_SIMPLE-INITFLAG). Once initialization is done i_initflag is set to “ ”, hence the else part will be executed.

```
       * Check DataSource validity
             case i_dsource.
                    when 'ZDS_FM_TEST'.
                    when others.
                           if 1 = 2. message e009(r3). endif.
* this is a typical log call. Please write every error message like this
        log_write 'E'                  "message type
                  'R3'                 "message class
                  '009'                "message number
                  i_dsource   "message variable 1
                  ' '.                 "message variable 2
        raise error_passed_to_mess_handler.
    endcase.
```

In this step it validates the Datasource. If I_DSOURCE is found to be something else that the actual Datasource it raise an error. This code make sure that this FM is isolated and can only be used for a particular Datasource i.e. `ZDS_FM_TEST`.

```
   append lines of i_t_select to s_s_if-t_select.


    s_s_if-requnr    = i_requnr.
    s_s_if-dsource = i_dsource.
    s_s_if-maxsize   = i_maxsize.

append lines of i_t_fields to s_s_if-t_fields.
```

I_T_SELECT gets input from S_S_IF_SIMPLE-T_SELECT(this system field get populated by value in the selection screen of the Infopackage or the range of the range of the delta relivant field, if we are running deltas). Above code fills the components of S_S_IF (defined static previously), so that this becomes global throughout the Function Module.

```
  else.                 "Initialization mode or data extraction ?

************************************************************************
* Data transfer: First Call      OPEN CURSOR + FETCH
*                Following Calls FETCH only
************************************************************************

* First data package -> OPEN CURSOR
    if s_counter_datapakid = 0.
```

This else part will be executed after initialization of the FM is done (i.e. when `i_initflag becomes` ""). Next line checks whether counter is set to 0 of not. It will be true in the 2nd run.

```
* Fill range tables BW will only pass down simple selection criteria
* of the type SIGN = 'I' and OPTION = 'EQ' or OPTION = 'BT'.
      loop at s_s_if-t_select into l_s_select where fieldnm = 'EBELN'.
        move-corresponding l_s_select to l_r_ebeln.
        append l_r_ebeln.
      endloop.

      loop at s_s_if-t_select into l_s_select where fieldnm = 'ERDAT'.
        move-corresponding l_s_select to l_r_erdat.
        append l_r_erdat.
      endloop.

       loop at s_s_if-t_select into l_s_select where fieldnm = 'ZDATE'.
        move-corresponding l_s_select to l_r_zdate.
        l_r_zdate-sign = 'I'.
        l_r_zdate-option = 'GE'.
        clear l_r_zdate-high.
        append l_r_zdate.
      endloop.
```

Here the data from s_s_if-t_select is passed to auxiliary selection criteria structure (l_s_select), which is then appended to the respective range tables defined earlier.

Point to notice here is, in the 3rd and also the last loop we specifically mentioning the sign and option fields of the range table. This loop will be active during delta loads. This loop will ensure that the selection can only fetch records having created of changed date greater than last run date (Pointer of the Generic Extractor will be shown in the next section). This L_R_ZDATE will again be used in the Open curser statement, in side where claws.

```
        open cursor with hold s_cursor for
       select ebeln
              erdat
              aedat
              loekz
              ntgew
              gewei
              from ztab_ds_fm_test
                   where ( ( erdat in l_r_zdate
                   or  aedat in l_r_zdate )
                   and ebeln in l_r_ebeln
                   and erdat in l_r_erdat ).
       endif.
```

Above code opens a curser to fetch data from database on basis of the given criteria. Find Cursor related details under Appendix.

Inside where claws we are checking whether ERDAT of the AEDAT in there with in the range criteria and select records accordingly. This will be considered as the first fetch statement.

Open cursor Statement will only be executed once during 2nd run, when the Packet Id is 0

```
          fetch next cursor s_cursor
              appending corresponding fields
              of table i_ztab_ds_fm_test
              package size s_s_if-maxsize.
```

Above code brings records in the form of packets in to internal table , where the no of records per packed is restricted to `s_s_if-maxsize`.

```
        if sy-subrc <> 0.
        close cursor s_cursor.
        raise no_more_data.
        else.
```

If in the FETCH NEXT CUSSOR wont returns any records then SY-SUBRC will automatically set to 4. Once above condition is fulfilled cursor is closed and No_MORE_DATA exception is raised.

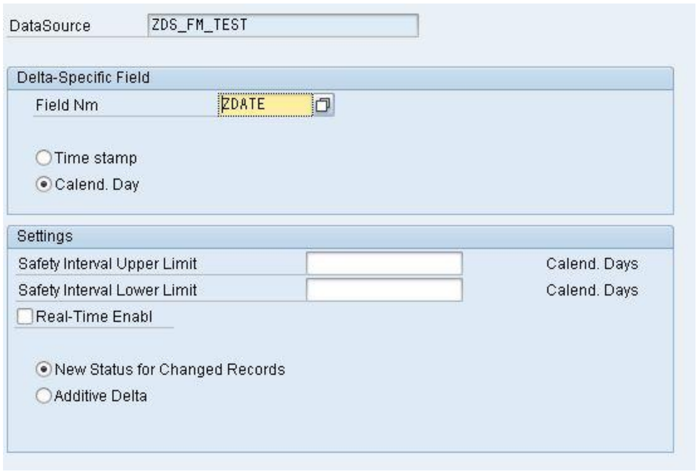
```
    if i_ztab_ds_fm_test[] is not initial.


             loop at i_ztab_ds_fm_test assigning <fs_x_ztab_ds_fm_test>.


                    if <fs_x_ztab_ds_fm_test>-aedat is initial.

                    move <fs_x_ztab_ds_fm_test>-erdat to w_t_data-zdate.
                   else.

                          move <fs_x_ztab_ds_fm_test>-aedat to w_t_data-zdate.
                   endif.

                   move:  <fs_x_ztab_ds_fm_test>-ebeln to w_t_data-ebeln,
                    <fs_x_ztab_ds_fm_test>-erdat to w_t_data-erdat,
                    <fs_x_ztab_ds_fm_test>-aedat to w_t_data-aedat,
                    <fs_x_ztab_ds_fm_test>-loekz to w_t_data-loekz,
                    <fs_x_ztab_ds_fm_test>-ntgew to w_t_data-ntgew,
                    <fs_x_ztab_ds_fm_test>-gewei to w_t_data-gewei.

                   append w_t_data to e_t_data.

                   endloop.
      endif.
```

Above code Checks whether internal table is having records or not. If yes , it populate the work area w_t_data which in turn appended to the output structure E_T_DATA. Here the ZDATE is filled with either AEDAT or ERDAT whichever is greater.

```
endif.
       s_counter_datapakid = s_counter_datapakid + 1.

       endif.              "Initialization mode or data extraction ?

endfunction
```

Here the data packet is incremented by one every time FM is called.

4. Activate the FM and then the Function Group.
5. Go to transaction RSO2, create a Transaction type Datasource, Provide the details like App. Component, Texts, and select Extraction by Function Module. Under Generic Delta tab, provide the delta specific field as ZDATE Select Cal. Day radio button. Save it and we are done.



DataSource ZDS_FM_TEST

Delta-Specific Field

Field Nm ZDATE

( ) Time stamp
(●) Calend. Day

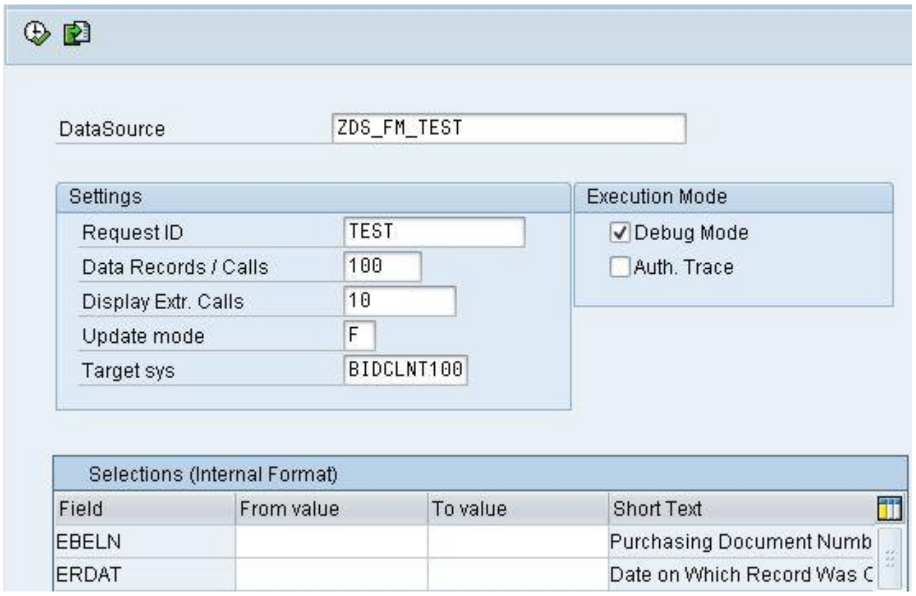
Settings

Safety Interval Upper Limit ____ Calend. Days
Safety Interval Lower Limit ____ Calend. Days
☐ Real-Time Enabl

(●) New Status for Changed Records
( ) Additive Delta

Internal Processing (Debugging)

If we check our extractor in RSA3, with debug mode switched on we can understand the Data Extraction process on our newly created Extractor.



DataSource: ZDS_FM_TEST

Settings

Request ID	TEST
Data Records / Calls	100
Display Extr. Calls	10
Update mode	F
Target sys	BIDCLNT100

Execution Mode

[✓] Debug Mode
[ ] Auth. Trace

Selections (Internal Format)

Field	From value	To value	Short Text
EBELN			Purchasing Document Numb
ERDAT			Date on Which Record Was C

Function RSA3_GET_DATA_SIMPLE is called first which in turn calls function S_FNAME containing the actual function module ZRSAX_BIW_GET_DATA_SIMPLE_TEST (our FM).

RSA3_GET_DATA_SIMPLE is the place from where our FM getting its value.

Snapshot of an assignment code inside function RSA3_GET_DATA_SIMPLE

```
EXPORTING      I_REQUNR                    = S_S_IF_SIMPLE-REQUNR
              I_DSOURCE                   = S_S_IF_SIMPLE-DSOURCE
              I_MAXSIZE                   = S_S_IF_SIMPLE-MAXSIZE
      I_INITFLAG                  = S_S_IF_SIMPLE-INITFLAG
TABLES
              I_T_SELECT                  = S_S_IF_SIMPLE-T_SELECT
              I_T_FIELDS                  = S_S_IF_SIMPLE-T_FIELDS
```

Where I_T_SELECT is the structure containing value from the standard table S_S_IF_SIMPLE-T_SELECT. S_S_IF_SIMPLE consists of

Requnr, dsource, maxsize, initflag, readonly, t_select, t_fields

I_T_SELECT contains the selection criteria defined at the info package level as well as it being used during Delta processing it gets populated with present date as High and one date after the last delta execution will become the Low value.

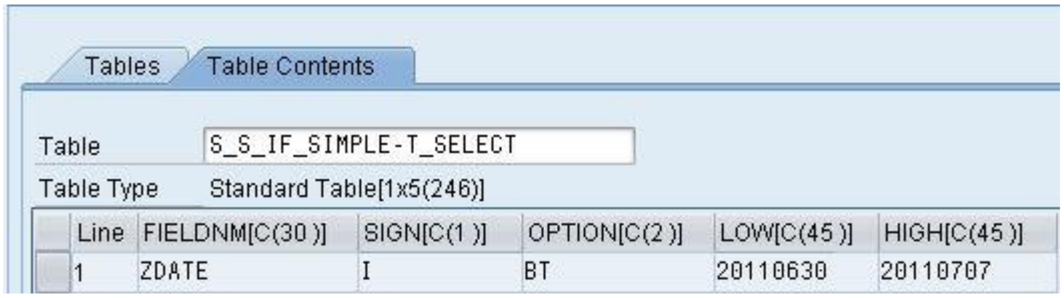


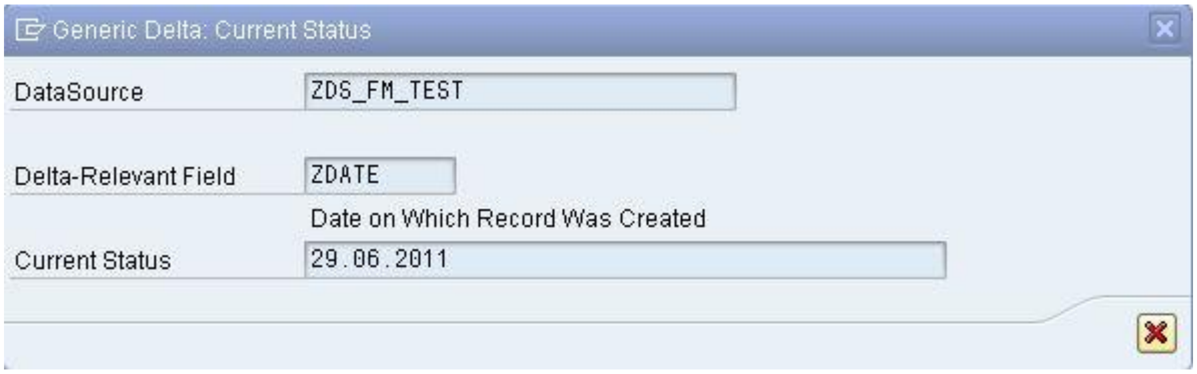
Table: S_S_IF_SIMPLE-T_SELECT

Table Type: Standard Table[1x5(246)]

Line	FIELDNM[C(30)]	SIGN[C(1)]	OPTION[C(2)]	LOW[C(45)]	HIGH[C(45)]
1	ZDATE	I	BT	20110630	20110707

Delta executed on 07-07-2011(20110707) and the date on which Delta was last executed 29-06-2011(20110629)

Snapshot of the Pointer of the Generic Extractor in RSA7:



DataSource	ZDS_FM_TEST
Delta-Relevant Field	ZDATE
	Date on Which Record Was Created
Current Status	29.06.2011

And I_T_FIELDS contains all the fields in the structure used as extract structure in the Data source which is transferred from Standard table S_S_IF_SIMPLE-T_FIELDS.

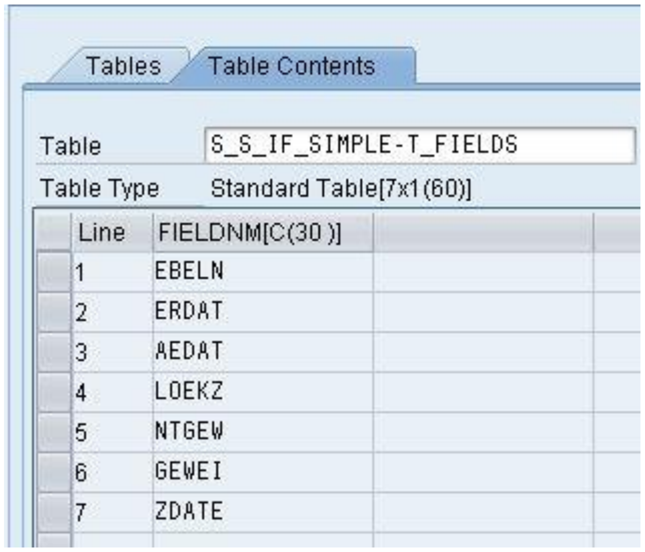


Table S_S_IF_SIMPLE-T_FIELDS

Table Type Standard Table[7x1(60)]

Line	FIELDNM[C(30)]
1	EBELN
2	ERDAT
3	AEDAT
4	LOEKZ
5	NTGEW
6	GEWEI
7	ZDATE

And the exception values are set as bellow

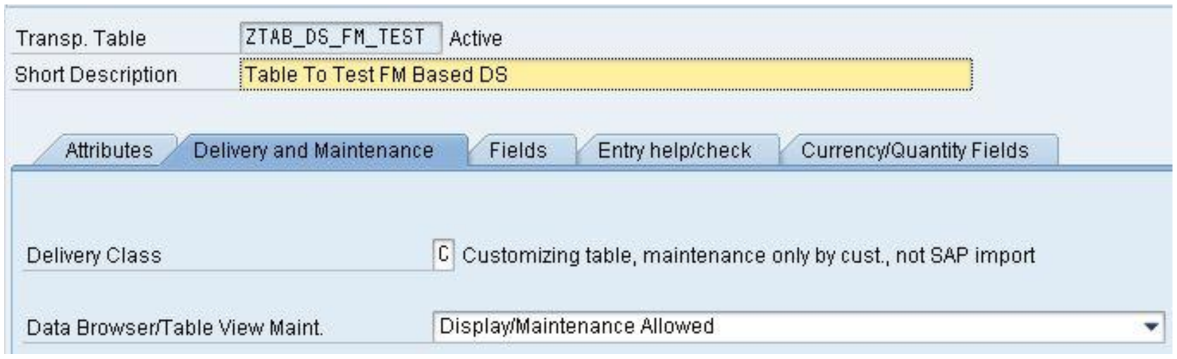
```
EXCEPTIONS
      NO_MORE_DATA                 = 1
      ERROR_PASSED_TO_MESS_HANDLER = 2
```

After first execution it will get back to the next program in the stack i.e. program for DATA_TRANSFER. No data is transferred in this cycle.

Auxiliary Steps

Steps for Creating Transparent Table

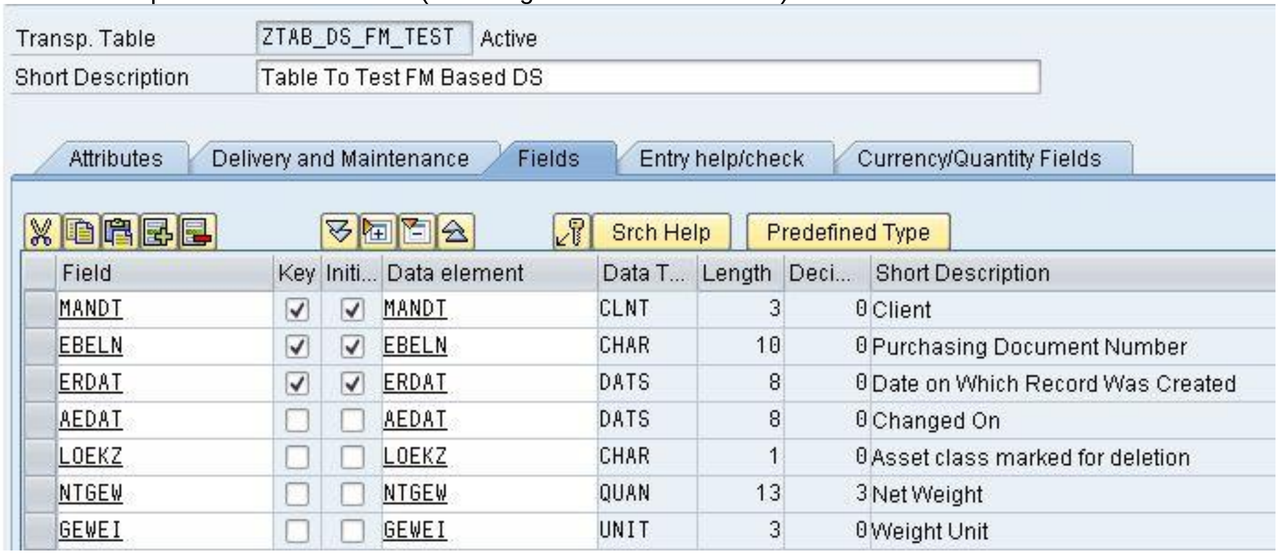
GO to TCode SE11 enter transparent table name (ZTAB_DS_FM_TEST). Enter *Short Description*, *Delivery Class* and *Data Browser/Table View Maint* as shown below.



Transp. Table	ZTAB_DS_FM_TEST	Active
Short Description	Table To Test FM Based DS	
Delivery Class	C	Customizing table, maintenance only by cust., not SAP import
Data Browser/Table View Maint.	Display/Maintenance Allowed	

Note: - Display/Maintenance Allowed : with this option we can insert records .

For this example we choose 7 fields (including AEDAT and ERDAT).

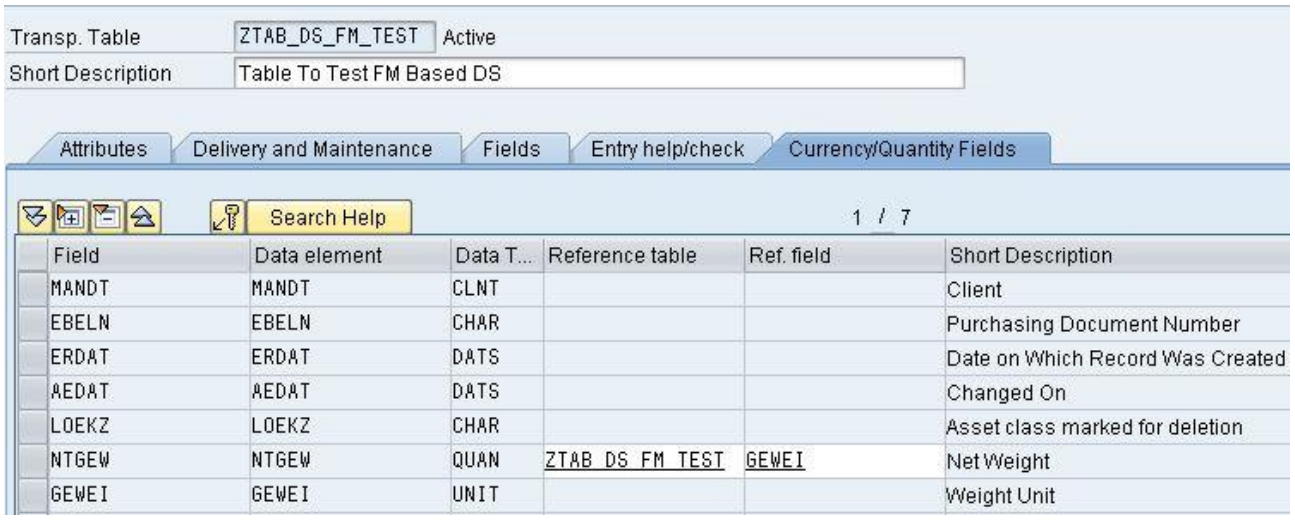


Transp. Table	ZTAB_DS_FM_TEST	Active
Short Description	Table To Test FM Based DS	

Field	Key	Initi...	Data element	Data T...	Length	Deci...	Short Description
MANDT	✓	✓	MANDT	CLNT	3	0	Client
EBELN	✓	✓	EBELN	CHAR	10	0	Purchasing Document Number
ERDAT	✓	✓	ERDAT	DATS	8	0	Date on Which Record Was Created
AEDAT	☐	☐	AEDAT	DATS	8	0	Changed On
LOEKZ	☐	☐	LOEKZ	CHAR	1	0	Asset class marked for deletion
NTGEW	☐	☐	NTGEW	QUAN	13	3	Net Weight
GEWEI	☐	☐	GEWEI	UNIT	3	0	Weight Unit

Note: - Client (MANDT) field is mandatory and it must be KEY including other primary key(s). In this scenario we took Purchasing Document Number (EBELN) and Date on Which Record Was Created(ERDAT) as KEY.

Enter the reference fields for Quantity and Currency fields used in the transparent table.



Transp. Table	ZTAB_DS_FM_TEST Active
Short Description	Table To Test FM Based DS

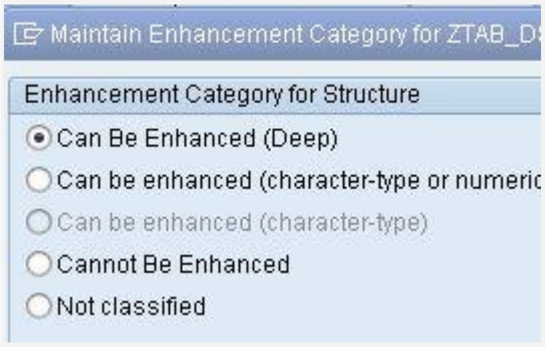
Attributes | Delivery and Maintenance | Fields | Entry help/check | Currency/Quantity Fields

Search Help 1 / 7

Field	Data element	Data T...	Reference table	Ref. field	Short Description
MANDT	MANDT	CLNT			Client
EBELN	EBELN	CHAR			Purchasing Document Number
ERDAT	ERDAT	DATS			Date on Which Record Was Created
AEDAT	AEDAT	DATS			Changed On
LOEKZ	LOEKZ	CHAR			Asset class marked for deletion
NTGEW	NTGEW	QUAN	ZTAB_DS_FM_TEST	GEWEI	Net Weight
GEWEI	GEWEI	UNIT			Weight Unit

For *Net Weight (NETGEW)* we took same table as reference table and *Weight Unit (GEWEI)* as *Ref Field.*

Note: - Set the *Enhancement Category* under tab *Extras* and set as *Can be enhanced (Deep).*

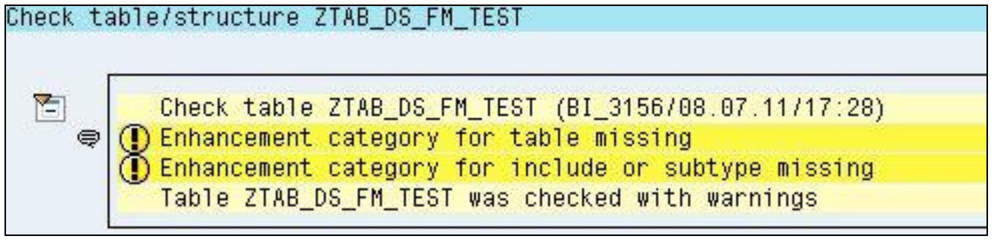


Maintain Enhancement Category for ZTAB_D

Enhancement Category for Structure

- [✓ Can Be Enhanced (Deep)]
- Can be enhanced (character-type or numeric
- Can be enhanced (character-type)
- Cannot Be Enhanced
- Not classified

We set it well in advance to avoid the warning message on activation of this Table.



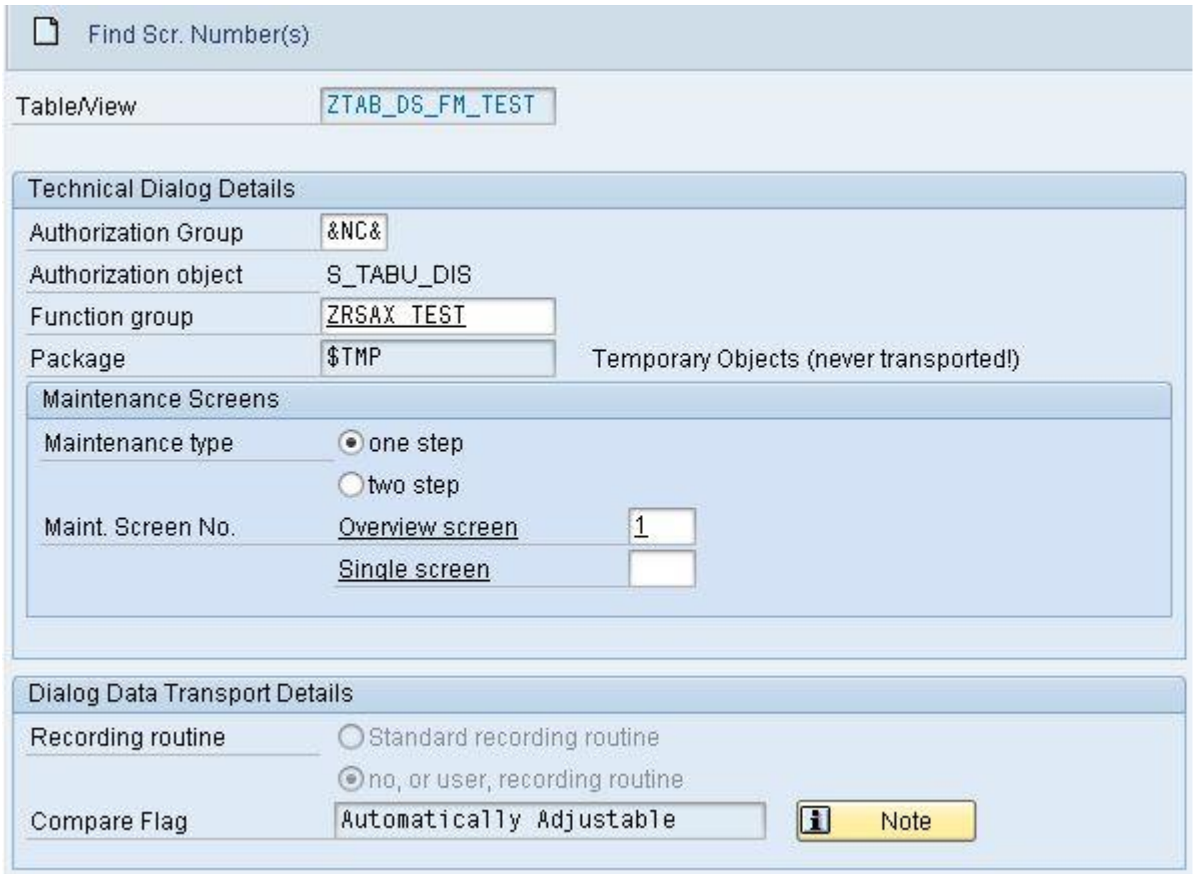
```
Check table/structure ZTAB_DS_FM_TEST

   Check table ZTAB_DS_FM_TEST (BI_3156/08.07.11/17:28)
 ! Enhancement category for table missing
 ! Enhancement category for include or subtype missing
   Table ZTAB_DS_FM_TEST was checked with warnings
```

Now activate the Table.

Create Table Maintenance Generator

Maintenance dialogs can be created for SAP or customer tables or views. Table maintenance activities (like insert, delete and update etc) can be carried out with Table Maintenance Dialog. Record maintenance is done from TCode SM30.

Go to tab *Utilities (M)* select *Table Maintenance Generator* enter parameters as below:



Table/View: ZTAB_DS_FM_TEST

Authorization Group: &NC&
Authorization object: S_TABU_DIS
Function group: ZRSAX_TEST
Package: $TMP Temporary Objects (never transported!)

Maintenance type: [✓ one step] two step
Maint. Screen No.: Overview screen 1
Single screen

Recording routine: Standard recording routine [✓ no, or user, recording routine]
Compare Flag: Automatically Adjustable

Authorization Group: &NC& (w/o auth.group)

Function group: Either enter a new Functional Group (and this will get created when we try to create the table Maintenance Dialog) or create a new Functional Group from TCode SE80.

Package: Assign prebuilt packager create one from SE80 (in this scenario we used local object *$TMP)*

Maintenance type : One Step and Over View Screen is auto suggested from the button *Find Scr. Number(s)* and then select Propose screen number(s).

Note: - Single step: Only overview screen is created i.e. the Table Maintenance Program will have only one screen where you can add, delete or edit records.

Two step: Two screens namely the overview screen and Single screen are created. The user can see the key fields in the first screen and can further go on to edit further details.

Press F6 (create).

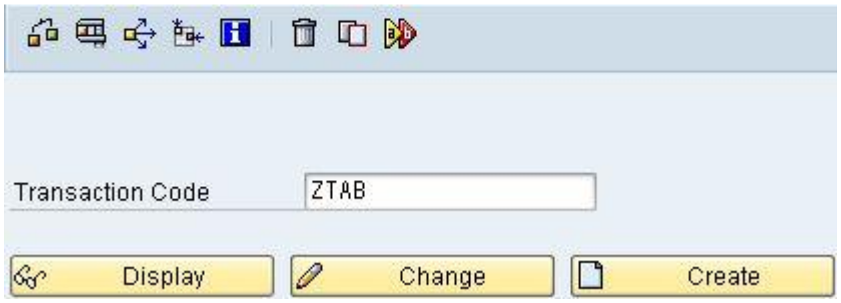
Now activate the Table.

We can get read of the hardship of Sm30's initial screen that we use for Table maintenance by creating Table/View specific TCode as below.

Create Tcode for the Table Maintenance Generator

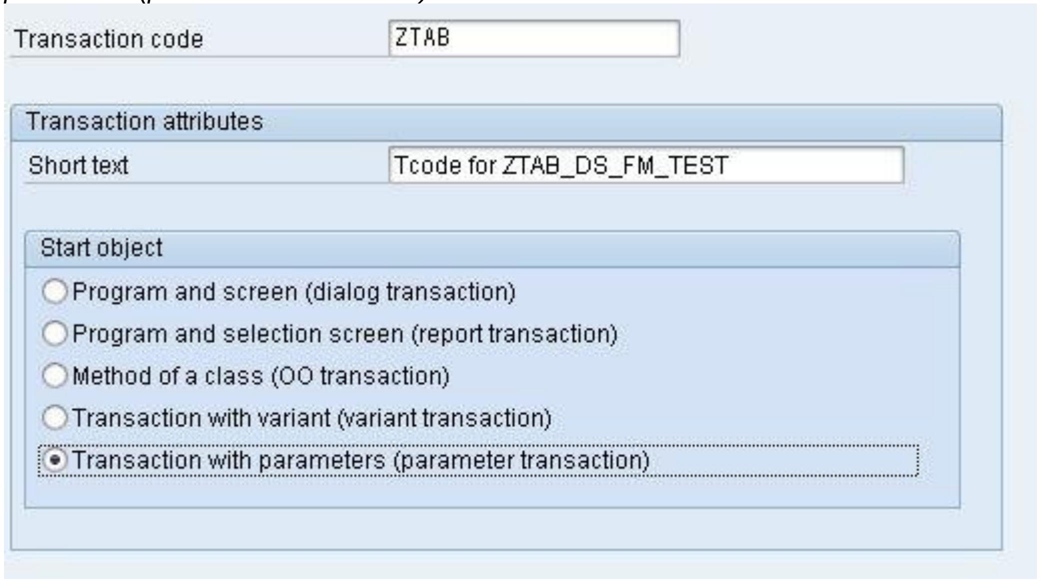
We can create custom TCode to directly access the table maintenance instead of using SM30.

Execute the Tcode SE93, enter the transaction code Name and click on Create Button.



Transaction Code: ZTAB

A popup will appear asking for the *Short text* and *Start object.* Select *Transaction with parameters(parameters transaction)*



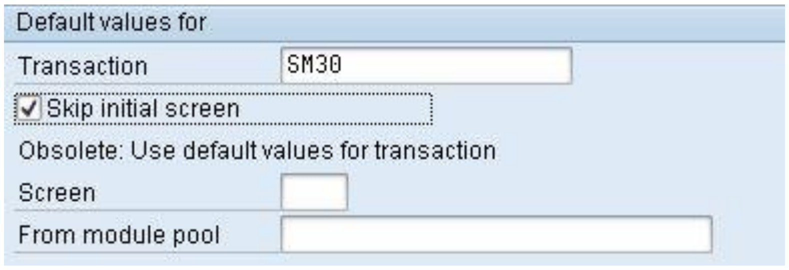
Transaction code: ZTAB

Short text: Tcode for ZTAB_DS_FM_TEST

Start object:
- ◯ Program and screen (dialog transaction)
- ◯ Program and selection screen (report transaction)
- ◯ Method of a class (OO transaction)
- ◯ Transaction with variant (variant transaction)
- ● Transaction with parameters (parameter transaction)

Note: - Parameter transactions allow you to pre assign values to the fields on the initial screen. If you supply all of the necessary entries for the initial screen in this way, you can suppress the screen when the transaction is executed.

Under *Default values of* for *Transaction* enter *SM30*. Check the *Skip initial screen* checkbox.



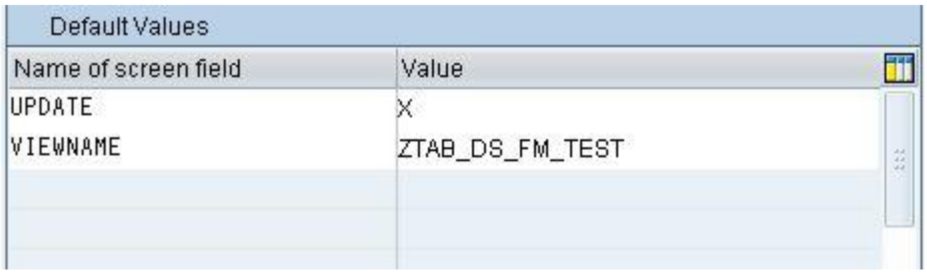
Transaction: SM30

☑ Skip initial screen

Screen:

From module pool:

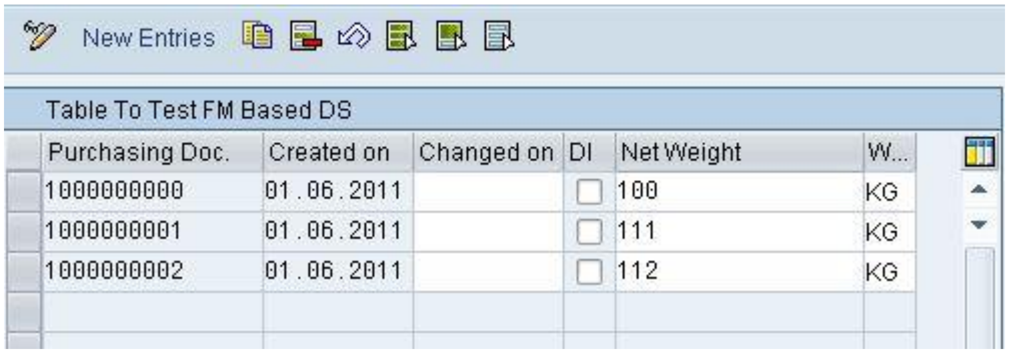
Now set the default screen fields of SM30 from *Default Values enter* the corresponding *Screen field name* and their default *value as below.*



Name of screen field	Value
UPDATE	X
VIEWNAME	ZTAB_DS_FM_TEST

Note: - Above default values will be passed during TCode call and field screen of SM30 will populated by the above values.

Now we have our own custom TCode for table maintenance of *ZTAB_DS_FM_TEST.* On execution it will take us directly to the maintenance screen of our Table.



Purchasing Doc.	Created on	Changed on	DI	Net Weight	W...
1000000000	01.06.2011		☐	100	KG
1000000001	01.06.2011		☐	111	KG
1000000002	01.06.2011		☐	112	KG

Appendix

STATICS:

Same as in other programming languages. The life of a variable with **STATICS** declared variables corresponds to the same one of a global data object. The variable is generated once when loading the framework program in the internal mode, and the contents set to the start value of the **VALUE** addition. Calling and ending the procedure have no effect on the life and content.

The statement **STATICS** for declaring static variables can only be used in subroutines, function modules, and static methods.

RANGE:

Data range variable is the type of variable where you can have a set range of value, the low and the high value. Sometimes we want to have a variable that's not an object such as Parameters or Select-Options, and use this variable as an input values into our select query where condition.

Range statement is equivalent to the following Select statement:

```
DATA: BEGIN OF rtab OCCURS {10|n},
        sign   TYPE c LENGTH 1,
        option TYPE c LENGTH 2,
        low    LIKE dobj,
        high   LIKE dobj,
      END OF rtab.
```

An internal table **`rtab`** with the structure of a selection table and a header line is declared. Without the addition **`OCCURS`** the initial memory requirement of the ranges-table is set to ten rows. With the addition **`OCCURS`**, you can specify a numeric literal or a numeric constant **`n`** to determine a different initial memory requirement.

While we use SELECT-OPTIONS system implicitly creates the select options internal table which contains the fields of SIGN, OPTION, LOW & HIGH. But in case of RANGES, this internal table should be defined explicitly.

FUNCTION MODULE

The initial value of an internal table is an empty table without lines.

Importing parameter:

These are export parameters that have to be supplied with data to the function module. They have to be supplied with data unless they are flagged as optional. We cannot change these parameter values in the function module. We have another type called changing parameters for them.

Exporting parameters:

The calling program receives result as export parameters from the function module.

Changing:

Changing parameters performs both the functions like importing and exporting data to and from the function module. The supplied variable can be changed in the function module and returned to the calling program.

Tables:

Table parameters are especially used to pass internal tables. They also function like changing parameters. They have to be supplied with values unless they are marked as optional.

Exceptions:

Exceptions are for handling situations where in which normal execution of program becomes difficult. For example if there are not entries in the table TAB1 that meet the selection criteria. We need NOT_FOUND exception for this.

CURSOR:

Database cursor

Pointer to the resulting set of a database selection. The database cursor is always assigned to a line of the resulting set. Cursor helps iterate over a collection of rows in the result set.

The CURSOR follows a three stage process of getting data from the database. The process is OPEN CURSOR...FETCH CURSOR...CLOSE CURSOR. The detailed usage of this process is described below.

```
OPEN CURSOR [WITH HOLD] <c> FOR SELECT      <result>
                                  FROM      <source>
                                  [WHERE    <condition>]
```

You can use all clauses of the SELECT statement apart from the INTO clause.

Caution: When we are using 'INTO CORRESPONDING FIELDS OF' in the FETCH statement of the CURSOR, it will display only the data for the latest SELECT statement, in this, the data is not appended but over-written.

```
FETCH NEXT CURSOR dbcur {INTO|APPENDING} target
```

The syntax and meaning of the addition INTO or APPENDING target are completely synonymous with the additions of the SELECT statement. If you specify non-table-type data objects after INTO, then one line is extracted. If an internal table is specified after INTO or APPENDING, then either all lines get extracted, or as many as specified in the addition PACKAGE SIZE.

The statement FETCH moves the position of the database cursor (which is linked to dbcur) by the amount of extracted lines to the next line to be extracted. If you extracted the last line of the resulting set in a FETCH statement, then every subsequent FETCH statement, in which dbcur is linked to the same database cursor, sets sy-subrc to 4 without influencing the data objects specified after INTO or APPENDING.

Use of cursor over select statement:

In a normal SELECT statement, the data is directly read into the target area specified by the INTO clause. When we use a SELECT...ENDSELECT statement, records are read row by row and individually access the RDBMS each and every time. Using a CURSOR decouples the process of INTO clause and the SELECT statement. After getting the complete data, the data is put into the target area. CURSOR creates a pointer to the DB record being created, with this; data is not actually copied into ABAP Application Server. CURSOR is mainly used to extract data in segments or chunks from the DB table. Conversely for the SELECT statement and its variants, data is directly copied into a data object in the ABAP memory. Generally the CURSOR is used in cases where we need to get data in chunks.

Related Content

Create a Maintenance Dialog

Function Module: Interface Description and Procedure

Cursor

For more information, visit the EDW homepage

Disclaimer and Liability Notice