FETCH and OPEN CURSOR Analysis

Applies to: R/3 4.6

Summary

Recently, I have been working with programs that extract huge amounts of data, for the purpose of BI. They often use the OPEN CURSOR / FETCH construct, to control the amount of records given to an "extractor" program prior, being sent to the BI system. The nature of some of these programs can require millions of records to be returned into an internal table and processed accordingly.

I have seen OPEN CURSOR / FETCH, but until now, not extensively, nor had I understood fully, the reasons why it should, or should not, be used.

Having searched the net for a simple explanation, and finding a couple of articles, but not really helping, I decided to perform some real tests myself and reach my own conclusions.

Author(s): Glen Spalding
Company: gingle
Created on: 21stMarch 2009

Author Bio

To date, I have worked with SAP in the technical area for over 13 years. I started as a Technical Consultant for one of the Implementation Partners in the UK, then became a contractor a few years after, working all over Europe. I gave up contracting in search for work in sunnier climates, which lands me here, in Australia, right now.

Although I am cross training myself into a Business Intelligence (BI) Role, I still find areas of ABAP challenging and powerful. This article demonstrates that ABAP, still to this day, easily accommodates future requirements.

Introduction

I must warn you now, this document does take a couple of reads before getting used to it.

Anyway, as my summary explains, I have recently been working with some SAP Extractor programs that retrieve large amounts of data, using the FETCH construct.

In search of knowledge, I ended up writing this document to explain a number of advanced concepts I found in such programs. Furthermore, I found myself extending the knowledge, to fully incorporate the use of parallel cursors and processes.

It is important to me to demonstrate the manner in which one would use a FETCH statement, and what benefit it can achieve. In doing so, I created a test program that measures the duration of numerous SELECT statements, as they are executed using different code.

I have also tried to limit the amount of in depth analysis so that this document serves as an initial platform for further investigation.

Test Program ZGTEST

When testing data retrieval, be mindful that test fields could be keys, or indexes, as this could yield conflicting results. Retrieving Keys or Index fields only, may not be representative of your requirement.

In my test program, you will see I am retrieving 5 fields of which some are not keys, nor indexed. Each SELECT statement contains a WHERE clause that utilizes an Index for the selection - visible in SQL Trace (ST05). Sufficient for my testing, but for specific testing, appropriate fields, and WHERE clauses, for selection will need to be used.

My test program contains the following:

Simple statements needed to only measure the data retrieval. Hence, the program on its own, pretty much does nothing.

Some simple fields used for outputting the chosen options, mode, number of records, and duration.

5 "checkbox" Options that determine which SELECT statements get executed for measuring the duration. Each SELECT statement can be identified by the WHERE clause. The 5th WHERE clause is programmed so it can be compared to the WHERE clauses 3 and 4 combined.
The SELECT statement extension, BYPASSING BUFFER is used, in an attempt to avoid measuring buffered records. What I am interested in is the retrieval of data from the database to the application server.

I have yet to experiment with the HOLD extension of the OPEN CURSOR statement.

**Options**

Throughout the document, you will hear me refer to the program Options. These are effectively the SELECT Statements. There are 5 Options.

1st WHERE clause 2009

The WHERE clause of the SELECT statement retrieves all records where GJHAR = 2009

2nd WHERE clause 2008

The WHERE clause of the SELECT statement retrieves all records where GJHAR = 2008

3rd WHERE clause 2007

The WHERE clause of the SELECT statement retrieves all records where GJHAR = 2007

4th WHERE clause 2006

The WHERE clause of the SELECT statement retrieves all records where GJHAR = 2006

5th WHERE clause 2006 andF 2007

The WHERE clause of the SELECT statement retrieves all records where GJHAR in (2006, 2007)

**Modes**

Throughout the documents, you will hear me refer to the program Modes. These are effectively the ABAP Code methods in which the SELECT statements is called. The six modes are as follows:

**Mode 1: Single cursor into work area**

SELECT / ENDSELECT construct into a work area.

```plaintext
SELECT (gt_fields)  
FROM (g_table)  
INTO gs_wa  
BYPASSING BUFFER  
WHERE (gt_where1).  
ADD 1 TO g_count.  
ENDSELECT.
```

**Mode 2: Single cursor into table**

SELECT construct into a table.

SELECT (gt_fields)
    FROM (g_table)
    INTO TABLE gt_1 BYPASSING BUFFER
    WHERE (gt_where1).

**Mode 3: Multi cursor, single process into work area**

OPEN CURSOR / FETCH constructs into a work area. Minimum code overhead is required to avoid an endless LOOP, and Cursor maintenance.

```
OPEN CURSOR l_c1 FOR
    SELECT (gt_fields)
    FROM (g_table)
    BYPASSING BUFFER
    WHERE (gt_where1).

    WHILE NOT l_c1 IS INITIAL.
        IF NOT l_c1 IS INITIAL.
            FETCH NEXT CURSOR l_c1 INTO gs_wa.
            IF sy-subrc EQ 0.
                ADD 1 TO g_count.
            ELSE.
                CLOSE CURSOR l_c1.
            ENDIF.
        ENDIF.
    ENDBHILE.
```

**Mode 4: Multi cursor, single process into table**

OPEN CURSOR / FETCH constructs into a table. Minimum code overhead is required to avoid an endless LOOP, and Cursor maintenance.
OPEN CURSOR l_c1 FOR
    SELECT (gt_fields)
    FROM (g_table)
    BYPASSING BUFFER
    WHERE (gt_where1).

WHILE NOT l_c1 IS INITIAL.

    IF NOT l_c1 IS INITIAL.
        FETCH NEXT CURSOR l_c1 INTO TABLE gt_1 PACKAGE SIZE p_pkg.
        IF sy-subrc EQ 0.
            DESCRIBE TABLE gt_1 LINES g_lines.
            ADD g_lines TO g_count.
            ELSE.
                CLOSE CURSOR l_c1.
        ENDIF.
    ENDIF.

ENDWHILE.

Mode 5: Multi cursor, multi process into work area

Identical as Mode 3, OPEN CURSOR / FETCH constructs into a work area, however each SELECT statement is called within its own RFCs starting in a new task, on a particular server group.

Program ZGSTEST Construct

CALL FUNCTION 'ZGSFETCH' STARTING NEW TASK 'ZGSC1'
    DESTINATION IN GROUP p_svrgp
    PERFORMING return_info ON END OF TASK
    EXPORTING
        i_mode      = 'W'
        i_pkg       = p_pkg
        i_tablename = g_table
        it_fields   = gt_fields
        it_where    = gt_where1
    EXCEPTIONS
        OTHERS       = 1.

Function Module ZGSFETCH Construct
OPEN CURSOR l_cursor FOR
  SELECT (it_fields)
  FROM (i_tablename)
  BYPASSING BUFFER
  WHERE (it_where).

WHILE NOT l_cursor IS INITIAL.
  FETCH NEXT CURSOR l_cursor INTO CORRESPONDING FIELDS OF ls_wa.
  IF sy-subrc EQ 0.
    ADD 1 TO e_lines.
  ELSE.
    CLOSE CURSOR l_cursor.
  ENDIF.
ENDWHILE.

**Mode 6: Multi cursor, multi process into table**

Identical as Mode 4, OPEN CURSOR / FETCH constructs into a table, however each SELECT statement is called within its own RFCs starting in a new task, on a particular server group.

**Program ZGSTEST Construct**

```
CALL FUNCTION 'ZGSFETCH'
  STARTING NEW TASK 'ZGSC1'
  DESTINATION IN GROUP p_svrgp
  PERFORMING return_info ON END OF TASK
  EXPORTING
    i_mode   = 'T'
    i_pkg    = p_pkg
    i_tablename = g_table
    it_fields = gt_fields
    it_where  = gt_where1
  EXCEPTIONS
    OTHERS   = 1.
```

**Function Module ZGSFETCH Construct**
OPEN CURSOR l_cursor FOR
SELECT (it_fields)
   FROM (i_tablename)
   BYPASSING BUFFER
   WHERE (it_where).

WHILE NOT l_cursor IS INITIAL.
   *simply overwrite the table
     FETCH NEXT CURSOR l_cursor
     INTO CORRESPONDING FIELDS OF TABLE lt_1 PACKAGE SIZE i_pkg.
     DESCRIBE TABLE lt_1 LINES l_lines.
     IF sy-subrc EQ 0.
       ADD l_lines TO e_lines.
     ELSE.
       CLOSE CURSOR l_cursor.
     ENDIF.
ENDWHILE.

Test scenario

My R/3 system contained table COEP, with large amounts of records

Test Data Metrics

<table>
<thead>
<tr>
<th>Total COEP</th>
<th>= 64 million records</th>
</tr>
</thead>
<tbody>
<tr>
<td>GJHAR Year = 2009</td>
<td>= 4.5m records</td>
</tr>
<tr>
<td>GJHAR Year = 2008</td>
<td>= 7.4m records</td>
</tr>
<tr>
<td>GJHAR Year = 2007</td>
<td>= 7.1m records</td>
</tr>
<tr>
<td>GJHAR Year = 2006</td>
<td>= 6.7m records</td>
</tr>
<tr>
<td>Total 2006, 2007, 2008, 2009</td>
<td>= 25.7m records</td>
</tr>
</tbody>
</table>

Testing

All tests have been conducted independently from each other. That is, they have not been run, simultaneously.

Naturally, each system, at a point in time, will have a variety of factors that may influence the results. E.g. CPU load, User Load, other DB Load, etc.

I repeat from above ...

"When testing data retrieval, be mindful that test fields could be keys, or indexes, as this could yield conflicting results. Retrieving Keys or Index fields only, may not be representative of your requirement.

In my test program, you will see I am retrieving 5 fields of which some are not keys, nor indexed. Each SELECT statement contains a WHERE clause that utilizes an Index for the selection - visible in SQL Trace (ST05). Sufficient for my testing, but for specific testing, appropriate fields and WHERE clauses, for selection will need to be used."
SQL Trace

When testing the SQL traces (ST05) I used a different system, with fewer records, so that the response would be faster, and I could execute the program in real time (not in background). Please be mindful of this when comparing durations from SQL traces with the Test durations throughout this document.

FETCH Operation

If analyzing the results from the SQL trace in ST05, note the number of Records returned during each Fetch. Consider each FETCH, a communication.

When retrieving a large number of fields (or large amount of data) for each record, the number of records returned, per FETCH/communication, would be less, than if it were to retrieve fewer fields (or fewer data) for each record. This is because each database communication has certain bandwidth in which to retrieve the records.

Hence why it is good practice to only retrieve fields we require (or minimum data) when programming SELECT statements. More records can be retrieved into a program, in a single Fetch/communication, and therefore, will limit the number of Fetches/communications between the program (application server) and database server.

As a general guideline, the less communicating with the database, the faster our program will be.

Look at the example below from the test program ZGSTEST using Option 1.

You can see that there are three FETCH/communications. The first FETCH returns 1083 records, the second FETCH, also returns 1083 records, and then final FETCH, returning the remaining 437.

Now, I modified the code to select all fields from COEP, and run again. The results are evident of what I am saying above.

Notice the greater number of FETCH/communications now required. This is because we can only return 88 records per FETCH/communication due to the increased number of fields.

Naturally, this will take longer, and consequently slow down the program.

PREPARE and OPEN Operations

Another interesting couple of points when looking at the SQL trace are the PREPARE and OPEN operations.

When the program code runs a simple SELECT statement as with Option 1, we can see above that the database prepares, opens, and fetches the records.

If the program is run again, immediately after, the database simply re-opens the cursor, and fetches the appropriate records. See Below.

Now, as I run the same SELECT statement, but this time with Option 3, in which we set up our own cursor, interesting results surface.

No SQL trace was written during the OPEN CURSOR statement. However, the SQL trace below, is the result of the
first entered loop of the WHILE clause, at the first FETCH NEXT CURSOR statement.

Notice 1083 records has been retrieved, but in our program, see below in debug, only 1 record is available. Also note the SY-DBCNT is 1.

As the code continued through the remaining logic, no SQL trace was written, naturally. Now, upon the next FETCH NEXT CURSOR statement, within the next WHILE loop, no further SQL statement was written, but notice the SY-DBCNT is now 2.

When does the next set of 1083 records get retrieved?
I put SQL trace on, until SY-DBCNT reached 1082, and tested what would happen next.

No SQL trace was written.

Not until the program needed to retrieve record 1084, did a new SQL trace get written, and the next set of 1083 records retrieved from the database server to the test program.

This test, clearly demonstrates the OPEN CURSOR and FETCH method influences the communication between the test program (ZGSTEST) and the database server.

To summarize, upon an initial request of a record, using the FETCH NEXT CURSOR, the program initiates the PREPARE, OPEN, and FETCH operation. Within the FETCH operation, the database provided the maximum amount of records it could fit in a single FETCH operation (1083 records). These records where subsequently provided to the program upon each FETCH NEXT CURSOR statement, without any further database communication.

Only when the program requested the next record (record 1084), outside the initial FETCH communication, was the next set of records (another 1083 records) retrieved from the database, and available to the program via the FETCH NEXT CURSOR.

This means, I have the capability to retrieve number of x records from the database server, into my program, and process the records accordingly. Should my SELECT statement still be able to return more records, BUT my processing no longer requires the records, I can simply exit the loop, close the cursor and end. This is clearly something that could be extremely useful heavy processing.

Tests

Each Mode is tested, with 4 of the Options checked.

Individual Options test may be useful for baseline comparisons with the following results.

All tests are conducted by executing the program in a background task. Upon completion, the program spool is viewed using Job Overview (SM37).

Even with the SELECT extension BYPASSING BUFFER, results can vary, so I would recommend performing multiple identical tests, and taking an average.
Mode 1: Single Cursor into Work Area

By nature of this Mode, the simple SELECT/ENDSELECT INTO work area, is utilizing a single cursor, and will be performed in its own single process.

The next SELECT statement will begin upon the completion of the previous SELECT statement.

The number of records returned is counted within the SELECT/ENDSELECT loop.

Mode 1 Result

Mode 2: Single Cursor into Table

Each SELECT statement is performing an ARRAY fetch, by way of using the SELECT extension, INTO TABLE. The SELECT statement is still utilizing a single cursor, and will be performed in its own single process.

The next SELECT statement will begin upon the completion of the previous SELECT statement.

The record count is calculated using the DESCRIBE TABLE command.

Mode 2 Result

Mode 3: Multiple Cursor, Single Process into Work Area

I now begin testing with multiple Cursors. Each SELECT statement is prepares its own Cursor using the OPEN CURSOR command.

The preparation of each SELECT statement does not return any records into the program.

After the last (4th) OPEN CURSOR/SELECT preparation, then, commences the actual data retrieval by way of the FETCH command.

The FETCH command is wrapped within a LOOP, in this case a WHILE loop.

The command FETCH NEXT CURSOR is responsible for retrieving the data into the program. In this mode, the Cursor's record is retrieved one at a time, into a work area, controlled by the WHILE loop.

The number of records returned is counted within the WHILE loop for each successful FETCH. Upon an unsuccessful FETCH, i.e. no more records, the cursor is CLOSED. Logic
within the program maintains the Loop and Cursor.

In this Mode, at the height of the program, there will be 4 Cursors addressing the same table, based their own SELECT statement. Some may argue this is parallel Cursor processing, as there are multiple Cursors open simultaneously, however, each Cursor can only be processed at a single time due to the nature of the program. So I will argue that it is not true parallel processing. That luxury will be demonstrated later.

Mode 3 Result

Mode 4: Multiple Cursor, Single Process into Table

As before, the Cursors are prepared with the SELECT statement, read for data retrieval. During the WHILE loop to FETCH each Cursor's set of data, I am now able to control the number of records returned into the program by way of the extension PACKAGE SIZE, of the FETCH statement.

So, in the example here, you can see a parameter "Package Size" which I have defaulted to 5000 (optimal use of size is something that will need to be tested, along with the intended "width" of the record).

While the program still contains open Cursors, each Cursor will retrieve data in 5000 record blocks. The first 5000 into the program will be retrieved from the 1st Cursor, then, during the same loop, another 5000 records will be retrieved from the 2nd Cursor, and so on. Should a Cursor have no more records, it is CLOSED and no longer used.

Note: As records are retrieved, they are not appended into the internal table, using the APPEND command. I am only interested in the number of lines returned from the CURSOR into the internal table. So for each Cursor I simply use the same internal table (FETCH NEXT CURSOR ... INTO TABLE gt_1) and count the lines of the internal table, adding it to a final total.

Mode 4 Result

Mode 5: Multiple Cursors, Multiple Processes into Work Area

Until now, I found the benefits from using the FETCH Statement and maintaining multiple cursors have not
really been worth the overhead.

The next two tests finally enable me to benefit from the FETCH functionality and truly demonstrate multiple Cursors in their own Process.

However, with such benefits, comes a complexity.

Consider what we need to "fire" an own process. Answer, an RFC enabled Function module, called in NEW TASK. This syntax is readily available on SAP Help and documented very well.

If you have been wondering why, in my program, I wrote all the SELECT statements using variables, you will see now.

The main program, ZGSTEST, creates "field", "table" name and "where" clause, variables for all the SELECT statements. In this test, these variables are to be passed as parameters, to newly created RFC Function Module ZGSFETCH. The code in ZGSFETCH, replicates that of program ZGSTEST. The only extra complexity lies where I use the same RFC for both Work Area, and Table use. Determined by parameter i_mode. "W" for Work Area, and "T" for Table.

Having passed the SELECT variables to each RFC as parameters, the RFC is called using STARTING NEW TASK. Each RFC call will then commence in a separate Dialog Process.

Now, we are in parallel mode using Dialog Processes. But notice I have a Logon Server Group parameter. We can specify a Server Group or, left blank, the default server group, maintained in RZ12 is used. This parameter enables me to use the extension DESTINATION IN GROUP.

By using this we are truly in Parallel Mode, with the ability to split the program into separate processes over multiple servers, and return back to the main program.

Because I want to return a value (e_lines) from the RFC started in a NEW TASK, I must use the PERFORMING ... ON END OF TASK extension to specify a form. In this form, the syntax RECEIVE RESULTS FROM is used to retrieve the RFC importing parameters back into the program.

WAIT UNTIL command suspends the program ZGSTEST, whilst the RFC STARTING IN NEW TASK goes off and does its thing. When the RFC STARTING IN NEW TASK completes, the program resumes with RECEIVE RESULTS FROM and continues.

To summarize, the SELECT statement will be called inside an RFC Function Module, using STARTING NEW TASK, so that a completely new Process is initiated. The parameters of the RFC will determine what the SELECT statement will perform. Results from the RFC are returned into the suspended ZGSTEST program, and upon RFC completion, the program ZGSTEST is resumed, and the RFC importing parameters are retrieved. The program ZGSTEST continues as normal.

In this test, as we initiated 4 parallel processes. The returned time is the duration in which the longest process took, positively exhibiting parallel processing.

Mode 5 Result

Compare this to the individual results of each - 100 + 179 + 183 + 148 = 610, and you can see the overhead is worth it. Individual test results
Mode 6: Multiple Cursors, Multiple Processes into Table

As with the previous test, this test, truly performs parallel Cursor processing. However the parameter "Package Size" has an important duty when selecting records as an ARRAY, via the INTO TABLE construct. At one point, I attempted retrieving millions of records, into a program to process, without using the Package Size (set to 0 or initial). What this did was attempt a retrieval of all records at once. I exceeded the program memory limit, and incurred a runtime exception error.

By managing the Package Size, I reduce program memory consumption and avoided the runtime error. The internal table I retrieved into was managed appropriately.

Mode 6 Result

The results, again, speak for themselves. Again, Compare this to the individual results of each - 91 + 160 + 159 + 153 = 563, and you can see the overhead is worth it.

For completeness, and to avoid unnecessary complexity, I have avoided management of Dialog Processes when calling RFCs using STARTING NEW TASK. If you are going to use this method, then you must manage the availability of Dialog Processes within your program. In the example above, if there were no more Dialog Processes available, or a communication error occurred calling the RFC, you must manage the EXCEPTIONS raised from the RFC call. Again SAP help is to hand and well documented.

Summary

In performing the tests above, I have satisfied my curiosity as to the use of OPEN CURSOR / FETCH and multiple cursors.

Naturally the quantity of data, retrieval (where clause), hardware, load etc, will all have various effects on performance and efficiency in the end. My tests above merely identify a need to test on a representative system to ultimately reach a final decision.

However, these simple tests go a long way to explain what is occurring under the SELECT statement and with the FETCH command.

To answer my question as to why I would use the OPEN CURSOR / FETCH statements, here they are:

- "To control/limit the number of records returned into a program from a SELECT statement"
- "To exit a SELECT statement prematurely"
- "To enable multiple cursors when retrieving data"
I trust some education was gained, and I look forward to hearing from you all.

Regards

Glen
glen@gingle.com

Appendix

Note, the Program ZGSTEST calls function module ZGSFETCH. The best I can do is provide you with the source code. You will have to build the function module as appropriate with the provided source code to get everything working as above.

Do you best to cut and copy into a program.

Program ZGSTEST

REPORT ZGSTEST.
TYPES: BEGIN OF ty_table,
  kokrs TYPE kokrs,
  belnr TYPE co_belnr,
  buzei TYPE co_buzei,
  objnr TYPE j_objnr,
  wtbbr TYPE wtgxxx,
END OF ty_table.
SELECTION-SCREEN BEGIN OF BLOCK opts WITH FRAME TITLE text-tt1.
  PARAMETERS:
    p_opt1 TYPE char1 AS CHECKBOX DEFAULT 'X',
    p_opt2 TYPE char1 AS CHECKBOX DEFAULT 'X',
    p_opt3 TYPE char1 AS CHECKBOX DEFAULT 'X',
    p_opt4 TYPE char1 AS CHECKBOX DEFAULT 'X',
    p_opt5 TYPE char1 AS CHECKBOX DEFAULT 'X'.
SELECTION-SCREEN END OF BLOCK opts.
SELECTION-SCREEN SKIP 1.
SELECTION-SCREEN BEGIN OF BLOCK mode WITH FRAME TITLE text-tt2.
  PARAMETERS:
    p_1 TYPE char1 RADIOBUTTON GROUP radi,
    p_2 TYPE char1 RADIOBUTTON GROUP radi,
    p_3 TYPE char1 RADIOBUTTON GROUP radi,
    p_4 TYPE char1 RADIOBUTTON GROUP radi,
    p_5 TYPE char1 RADIOBUTTON GROUP radi,
    p_6 TYPE char1 RADIOBUTTON GROUP radi,
    p_pkg TYPE i DEFAULT 5000,
    p_svrgp TYPE rzlli_apcl.
SELECTION-SCREEN END OF BLOCK mode.
* working variables
DATA:
  g_mode TYPE i,
gs_wa  TYPE ty_table,          "#EC NEEDED
  g_count  TYPE i,
  g_time  TYPE i,
  g_lines  TYPE i,
  gt_1    TYPE STANDARD TABLE OF ty_table,
* select statement variables
  gt_fields  TYPE wheretab,
  g_table   TYPE char30,
  gt_where1  TYPE wheretab,
  gt_where2  TYPE wheretab,
  gt_where3  TYPE wheretab,
  gt_where4  TYPE wheretab,
  gt_where5  TYPE wheretab.
START-OF-SELECTION.
*initialize
  FREE: g_mode, gs_wa, gt_fields, g_count, g_table, g_time, g_lines,
  gt_1, gt_fields, g_table,
  gt_where1, gt_where2, gt_where3, gt_where4, gt_where5.
---------------------------------------------------------------------
*SELECT field selection
  APPEND 'KOKRS'  TO gt_fields.
  APPEND 'BELNR'  TO gt_fields.
  APPEND 'BUZEI'  TO gt_fields.
  APPEND 'OBJNR'  TO gt_fields.
  APPEND 'WTGBTR' TO gt_fields.
  FROM table
  g_table = 'COEP'.
  WHERE clause
  APPEND 'gjahr = ''2009'''  TO gt_where1.
  APPEND 'gjahr = ''2008'''  TO gt_where2.
  APPEND 'gjahr = ''2007'''  TO gt_where3.
  APPEND 'gjahr = ''2006'''  TO gt_where4.
  APPEND 'gjahr in (''2006'', ''2007'')'  TO gt_where5.
--------------
*do option on screen
  CASE 'X'.
    WHEN p_1.
      PERFORM do_single_cur_wa.  " single curs, into work area
    WHEN p_2.
      PERFORM do_single_cur_table.  " single curs, into table
    WHEN p_3.
      PERFORM do_multi_curs_wa.  " multi curs single Proc. wa
    WHEN p_4.
      PERFORM do_multi_curs_table.  " multi curs single Proc. tab
    WHEN p_5.
      PERFORM do_multi_curs_mp_wa.  " multi curs, multi Proc. wa
    WHEN p_6.
      PERFORM do_multi_curs_mp_table.  " multi curs, multi Proc. tab
  ENDCASE.
---------------------------------------------------------------------
END-OF-SELECTION.
DIVIDE g_time BY 1000000.  " convert to seconds
WRITE:/ 'Option 1 ', p_opt1,
/ 'Option 2 ', p_opt2,
/ 'Option 3 ', p_opt3,
/ 'Option 4 ', p_opt4,
/ 'Option 5 ', p_opt5.
SKIP.
WRITE:/ 'Mode ', g_mode,
/ 'Record Count ', g_count,
/ 'Time ', g_time, 'Seconds'.
* ---------------------------------------------------------------------
* FORM do_single_cur_wa
* ---------------------------------------------------------------------
FORM do_single_cur_wa.
g_mode = 1. " for displaying what mode was run, at end
GET RUN TIME FIELD g_time.
IF p_opt1 EQ 'X'.
SELECT (gt_fields)
FROM (g_table)
  INTO gs_wa
  BYPASSING BUFFER
  WHERE (gt_where1).
  ADD 1 TO g_count.
ENDSELECT.
ENDIF.
IF p_opt2 EQ 'X'.
SELECT (gt_fields)
FROM (g_table)
  INTO gs_wa
  BYPASSING BUFFER
  WHERE (gt_where2).
  ADD 1 TO g_count.
ENDSELECT.
ENDIF.
IF p_opt3 EQ 'X'.
SELECT (gt_fields)
FROM (g_table)
  INTO gs_wa
  BYPASSING BUFFER
  WHERE (gt_where3).
  ADD 1 TO g_count.
ENDSELECT.
ENDIF.
IF p_opt4 EQ 'X'.
SELECT (gt_fields)
FROM (g_table)
  INTO gs_wa
  BYPASSING BUFFER
  WHERE (gt_where4).
  ADD 1 TO g_count.
ENDSELECT.
ENDIF.
IF p_opt5 EQ 'X'.


SELECT (gt_fields)
    FROM (g_table)
    INTO gs_wa
    BYPASSING BUFFER
    WHERE (gt_where5).
    ADD 1 TO g_count.
ENDSELECT.
ENDIF.
GET RUN TIME FIELD g_time.
ENDFORM.

*---------------------------------------------------------------------
*       FORM do_single_cur_table
*---------------------------------------------------------------------
*       ........
*---------------------------------------------------------------------

FORM do_single_cur_table.
  g_mode = 2.  " for displaying what mode was run, at end
  GET RUN TIME FIELD g_time.
  IF p_opt1 EQ 'X'.
    SELECT (gt_fields)
       FROM (g_table)
       INTO TABLE gt_1 BYPASSING BUFFER
       WHERE (gt_where1).
    DESCRIBE TABLE gt_1 LINES g_lines.
    ADD g_lines TO g_count.
  ENDIF.
  IF p_opt2 EQ 'X'.
    SELECT (gt_fields)
       FROM (g_table)
       INTO TABLE gt_1 BYPASSING BUFFER
       WHERE (gt_where2).
    DESCRIBE TABLE gt_1 LINES g_lines.
    ADD g_lines TO g_count.
  ENDIF.
  IF p_opt3 EQ 'X'.
    SELECT (gt_fields)
       FROM (g_table)
       INTO TABLE gt_1 BYPASSING BUFFER
       WHERE (gt_where3).
    DESCRIBE TABLE gt_1 LINES g_lines.
    ADD g_lines TO g_count.
  ENDIF.
  IF p_opt4 EQ 'X'.
    SELECT (gt_fields)
       FROM (g_table)
       INTO TABLE gt_1 BYPASSING BUFFER
       WHERE (gt_where4).
    DESCRIBE TABLE gt_1 LINES g_lines.
    ADD g_lines TO g_count.
  ENDIF.
  IF p_opt5 EQ 'X'.
    SELECT (gt_fields)
       FROM (g_table)
INTO TABLE gt_1 BYPASSING BUFFER
WHERE (gt_where5).
DESCRIBE TABLE gt_1 LINES g_lines.
ADD g_lines TO g_count.
ENDIF.
GET RUN TIME FIELD g_time.
ENDFORM.

* -----------------------------
*     FORM do_multi_curs_wa
*-----------------------------
*     .......
*-----------------------------

FORM do_multi_curs_wa.

  g_mode = 3. " for displaying what mode was run, at end
DATA:
  1_c1 TYPE cursor,
  1_c2 TYPE cursor,
  1_c3 TYPE cursor,
  1_c4 TYPE cursor,
  1_c5 TYPE cursor.
GET RUN TIME FIELD g_time.
IF p_opt1 EQ 'X'.
  OPEN CURSOR 1_c1 FOR
    SELECT (gt_fields)
    FROM (g_table)
    BYPASSING BUFFER
    WHERE (gt_where1).
ENDIF.
IF p_opt2 EQ 'X'.
  OPEN CURSOR 1_c2 FOR
    SELECT (gt_fields)
    FROM (g_table)
    BYPASSING BUFFER
    WHERE (gt_where2).
ENDIF.
IF p_opt3 EQ 'X'.
  OPEN CURSOR 1_c3 FOR
    SELECT (gt_fields)
    FROM (g_table)
    BYPASSING BUFFER
    WHERE (gt_where3).
ENDIF.
IF p_opt4 EQ 'X'.
  OPEN CURSOR 1_c4 FOR
    SELECT (gt_fields)
    FROM (g_table)
    BYPASSING BUFFER
    WHERE (gt_where4).
ENDIF.
IF p_opt5 EQ 'X'.
  OPEN CURSOR 1_c5 FOR
    SELECT (gt_fields)
    FROM (g_table)
BYPASSING BUFFER
    WHERE (gt_where5).
ENDIF.
WHILE NOT l_c1 IS INITIAL OR
    NOT l_c2 IS INITIAL OR
    NOT l_c3 IS INITIAL OR
    NOT l_c4 IS INITIAL OR
    NOT l_c5 IS INITIAL.
    IF NOT l_c1 IS INITIAL.
        FETCH NEXT CURSOR l_c1 INTO gs_wa.
        IF sy-subrc EQ 0.
            ADD 1 TO g_count.
        ELSE.
            CLOSE CURSOR l_c1.
        ENDIF.
    ENDIF.
    IF NOT l_c2 IS INITIAL.
        FETCH NEXT CURSOR l_c2 INTO gs_wa.
        IF sy-subrc EQ 0.
            ADD 1 TO g_count.
        ELSE.
            CLOSE CURSOR l_c2.
        ENDIF.
    ENDIF.
    IF NOT l_c3 IS INITIAL.
        FETCH NEXT CURSOR l_c3 INTO gs_wa.
        IF sy-subrc EQ 0.
            ADD 1 TO g_count.
        ELSE.
            CLOSE CURSOR l_c3.
        ENDIF.
    ENDIF.
    IF NOT l_c4 IS INITIAL.
        FETCH NEXT CURSOR l_c4 INTO gs_wa.
        IF sy-subrc EQ 0.
            ADD 1 TO g_count.
        ELSE.
            CLOSE CURSOR l_c4.
        ENDIF.
    ENDIF.
    IF NOT l_c5 IS INITIAL.
        FETCH NEXT CURSOR l_c5 INTO gs_wa.
        IF sy-subrc EQ 0.
            ADD 1 TO g_count.
        ELSE.
            CLOSE CURSOR l_c5.
        ENDIF.
    ENDIF.
ENDWHILE.
GET RUN TIME FIELD g_time.
ENDFORM.
*   ---------------------------------------------------------------------
*   FORM do_multi_curs_table                                          *
*   ---------------------------------------------------------------------
FORM do_multi_curs_table.
g_mode = 4.  " for displaying what mode was run, at end
DATA:
l_c1 TYPE cursor,
l_c2 TYPE cursor,
l_c3 TYPE cursor,
l_c4 TYPE cursor,
l_c5 TYPE cursor.
GET RUN TIME FIELD g_time.
IF p_opt1 EQ 'X'.
  OPEN CURSOR l_c1 FOR
  SELECT (gt_fields)
  FROM (g_table)
  BYPASSING BUFFER
  WHERE (gt_where1).
ENDIF.
IF p_opt2 EQ 'X'.
  OPEN CURSOR l_c2 FOR
  SELECT (gt_fields)
  FROM (g_table)
  BYPASSING BUFFER
  WHERE (gt_where2).
ENDIF.
IF p_opt3 EQ 'X'.
  OPEN CURSOR l_c3 FOR
  SELECT (gt_fields)
  FROM (g_table)
  BYPASSING BUFFER
  WHERE (gt_where3).
ENDIF.
IF p_opt4 EQ 'X'.
  OPEN CURSOR l_c4 FOR
  SELECT (gt_fields)
  FROM (g_table)
  BYPASSING BUFFER
  WHERE (gt_where4).
ENDIF.
IF p_opt5 EQ 'X'.
  OPEN CURSOR l_c5 FOR
  SELECT (gt_fields)
  FROM (g_table)
  BYPASSING BUFFER
  WHERE (gt_where5).
ENDIF.
WHILE NOT l_c1 IS INITIAL OR
  NOT l_c2 IS INITIAL OR
  NOT l_c3 IS INITIAL OR
  NOT l_c4 IS INITIAL OR
  NOT l_c5 IS INITIAL.
IF NOT l_c1 IS INITIAL.
FETCH NEXT CURSOR l_c1 INTO TABLE gt_1 PACKAGE SIZE p_pkg.
IF sy-subrc EQ 0.
   DESCRIBE TABLE gt_1 LINES g_lines.
   ADD g_lines TO g_count.
ELSE.
   CLOSE CURSOR l_c1.
ENDIF.
ENDIF.
IF NOT l_c2 IS INITIAL.
   FETCH NEXT CURSOR l_c2 INTO TABLE gt_1 PACKAGE SIZE p_pkg.
   IF sy-subrc EQ 0.
      DESCRIBE TABLE gt_1 LINES g_lines.
      ADD g_lines TO g_count.
   ELSE.
      CLOSE CURSOR l_c2.
   ENDIF.
ENDIF.
IF NOT l_c3 IS INITIAL.
   FETCH NEXT CURSOR l_c3 INTO TABLE gt_1 PACKAGE SIZE p_pkg.
   IF sy-subrc EQ 0.
      DESCRIBE TABLE gt_1 LINES g_lines.
      ADD g_lines TO g_count.
   ELSE.
      CLOSE CURSOR l_c3.
   ENDIF.
ENDIF.
IF NOT l_c4 IS INITIAL.
   FETCH NEXT CURSOR l_c4 INTO TABLE gt_1 PACKAGE SIZE p_pkg.
   IF sy-subrc EQ 0.
      DESCRIBE TABLE gt_1 LINES g_lines.
      ADD g_lines TO g_count.
   ELSE.
      CLOSE CURSOR l_c4.
   ENDIF.
ENDIF.
IF NOT l_c5 IS INITIAL.
   FETCH NEXT CURSOR l_c5 INTO TABLE gt_1 PACKAGE SIZE p_pkg.
   IF sy-subrc EQ 0.
      DESCRIBE TABLE gt_1 LINES g_lines.
      ADD g_lines TO g_count.
   ELSE.
      CLOSE CURSOR l_c5.
   ENDIF.
ENDIF.
ENDWHILE.
GET RUN TIME FIELD g_time.
ENDFORM.
*---------------------------------------------------------------------
* FORM do_multi_curs_mp_wa                                      *
*---------------------------------------------------------------------
* ..........                                                  *
*---------------------------------------------------------------------
FORM do_multi_curs_mp_wa.
g_mode = 5. " for displaying what mode was run, at end
GET RUN TIME FIELD g_time.
IF p_opt1 EQ 'X'.
CALL FUNCTION 'ZGSFETCH' STARTING NEW TASK 'ZGSC1'
 DESTINATION IN GROUP p_svrgp
 PERFORMING return_info ON END OF TASK
 EXPORTING
   i_mode = 'W'
   i_pkg = p_pkg
   i_tablename = g_table
   it_fields = gt_fields
   it_where = gt_where1
 EXCEPTIONS
   OTHERS = 1.
ENDIF.
IF p_opt2 EQ 'X'.
CALL FUNCTION 'ZGSFETCH' STARTING NEW TASK 'ZGSC2'
 DESTINATION IN GROUP p_svrgp
 PERFORMING return_info ON END OF TASK
 EXPORTING
   i_mode = 'W'
   i_pkg = p_pkg
   i_tablename = g_table
   it_fields = gt_fields
   it_where = gt_where2
 EXCEPTIONS
   OTHERS = 1.
ENDIF.
IF p_opt3 EQ 'X'.
CALL FUNCTION 'ZGSFETCH' STARTING NEW TASK 'ZGSC3'
 DESTINATION IN GROUP p_svrgp
 PERFORMING return_info ON END OF TASK
 EXPORTING
   i_mode = 'W'
   i_pkg = p_pkg
   i_tablename = g_table
   it_fields = gt_fields
   it_where = gt_where3
 EXCEPTIONS
   OTHERS = 1.
ENDIF.
IF p_opt4 EQ 'X'.
CALL FUNCTION 'ZGSFETCH' STARTING NEW TASK 'ZGSC4'
 DESTINATION IN GROUP p_svrgp
 PERFORMING return_info ON END OF TASK
 EXPORTING
   i_mode = 'W'
   i_pkg = p_pkg
   i_tablename = g_table
   it_fields = gt_fields
   it_where = gt_where4
 EXCEPTIONS
   OTHERS = 1.
IF p_opt5 EQ 'X'.
    CALL FUNCTION 'ZGSFETCH' STARTING NEW TASK 'ZGSC5'
    DESTINATION IN GROUP p_svrgp
    PERFORMING return_info ON END OF TASK
    EXPORTING
        i_mode = 'W'
        i_pkg  = p_pkg
        i_tablename = g_table
        it_fields = gt_fields
        it_where  = gt_where5
    EXCEPTIONS
        OTHERS = 1.
ENDIF.

DATA: l_wait_flag.
WAIT UNTIL l_wait_flag = 'X'.
GET RUN TIME FIELD g_time.
ENDFORM.

*---------------------------------------------------------------------
*       FORM do_multi_curs_mp_table                                *
*---------------------------------------------------------------------
*       ........                                                   *
*---------------------------------------------------------------------

FORM do_multi_curs_mp_table.
    g_mode = 6.  " for displaying what mode was run, at end
    GET RUN TIME FIELD g_time.
    IF p_opt1 EQ 'X'.
        CALL FUNCTION 'ZGSFETCH'
            STARTING NEW TASK 'ZGSC1'
            DESTINATION IN GROUP p_svrgp
            PERFORMING return_info ON END OF TASK
            EXPORTING
                i_mode = 'T'
                i_pkg  = p_pkg
                i_tablename = g_table
                it_fields = gt_fields
                it_where  = gt_where1
            EXCEPTIONS
                OTHERS = 1.
        ENDIF.
    IF p_opt2 EQ 'X'.
        CALL FUNCTION 'ZGSFETCH' STARTING NEW TASK 'ZGSC2'
            DESTINATION IN GROUP p_svrgp
            PERFORMING return_info ON END OF TASK
            EXPORTING
                i_mode = 'T'
                i_pkg  = p_pkg
                i_tablename = g_table
                it_fields = gt_fields
                it_where  = gt_where2
            EXCEPTIONS
                OTHERS = 1.
        ENDIF.
    ENDIF.
IF p_opt3 EQ 'X'.
CALL FUNCTION 'ZGSFETCH' STARTING NEW TASK 'ZGSC3'
   DESTINATION IN GROUP p_svrgp
   PERFORMING return_info ON END OF TASK
   EXPORTING
      i_mode = 'T'
      i_pkg = p_pkg
      i_tablename = g_table
      it_fields = gt_fields
      it_where = gt_where3
   EXCEPTIONS
      OTHERS = 1.
ENDIF.
IF p_opt4 EQ 'X'.
CALL FUNCTION 'ZGSFETCH' STARTING NEW TASK 'ZGSC4'
   DESTINATION IN GROUP p_svrgp
   PERFORMING return_info ON END OF TASK
   EXPORTING
      i_mode = 'T'
      i_pkg = p_pkg
      i_tablename = g_table
      it_fields = gt_fields
      it_where = gt_where4
   EXCEPTIONS
      OTHERS = 1.
ENDIF.
IF p_opt5 EQ 'X'.
CALL FUNCTION 'ZGSFETCH' STARTING NEW TASK 'ZGSC5'
   DESTINATION IN GROUP p_svrgp
   PERFORMING return_info ON END OF TASK
   EXPORTING
      i_mode = 'T'
      i_pkg = p_pkg
      i_tablename = g_table
      it_fields = gt_fields
      it_where = gt_where5
   EXCEPTIONS
      OTHERS = 1.
ENDIF.
DATA: l_wait_flag.
WAIT UNTIL l_wait_flag = 'X'.
GET RUN TIME FIELD g_time.
ENDFORM.

* ---------------------------------------------------------------
*     FORM return_info           *
* ---------------------------------------------------------------
*        ........            *
* ---------------------------------------------------------------
* --> TASKNAME            *
* ---------------------------------------------------------------

FORM return_info USING taskname.
   RECEIVE RESULTS FROM
   FUNCTION 'ZGSFETCH'
IMPORTING
e_lines = g_lines.
ADD g_lines TO g_count.
ENDFORM.

**Function Module**

Don't forget to RFC enable the Function Module
FUNCTION zgfetch.
**"―---------------------------------------------------------------------------**
**"Local interface:
**" IMPORTING
**"     VALUE(I_PKG) TYPE I OPTIONAL
**"     VALUE(I_TABLENAME) TYPE CHAR30
**"     VALUE(IT_FIELDS) TYPE WHERETAB
**"     VALUE(IT_WHERE) TYPE WHERETAB
**"     VALUE(I_MODE) TYPE CHAR1 DEFAULT 'T'
**" EXPORTING
**"     VALUE(E_LINES) TYPE I
**"―---------------------------------------------------------------------------
DATA:
   l_lines   TYPE i,
   lt_1      TYPE STANDARD TABLE OF coep,
   ls_wa     TYPE coep,
   l_cursor  TYPE cursor.
OPEN CURSOR l_cursor FOR
SELECT (it_fields)
FROM (i_tablename)
BYPASSING BUFFER
WHERE (it_where).
* put the WHILE inside the mode check so it only checks once
IF i_mode EQ 'T'. " table
   WHILE NOT l_cursor IS INITIAL.
* simply overwrite the table, we are only interested in the numbers
   FETCH NEXT CURSOR l_cursor
   INTO CORRESPONDING FIELDS OF TABLE lt_1 PACKAGE SIZE i_pkg.
   DESCRIBE TABLE lt_1 LINES l_lines.
   IF sy-subrc EQ 0.
      ADD l_lines TO e_lines.
   ELSE.
      CLOSE CURSOR l_cursor.
   ENDIF.
ENDIF.
ENDWHILE.
ELSEIF i_mode EQ 'W'.
   WHILE NOT l_cursor IS INITIAL.
   FETCH NEXT CURSOR l_cursor INTO CORRESPONDING FIELDS OF ls_wa.
   IF sy-subrc EQ 0.
      ADD 1 TO e_lines.
   ELSE.
      CLOSE CURSOR l_cursor.
   ENDIF.
ENDIF.
ENDWHILE.
ENDIF.
FREE: ls_wa, lt_1.
ENDFUNCTION.