SAP ABAP Performance Tuning

Agenda

1 Definition

2 Techniques

3 Tools

What is Performance Tuning

An Efficient Program:Definition

- An efficient ABAP program is one which delivers the required output to the user in a finite time as per the complexity of the program

Advantages

- A performance optimized ABAP program saves the time of the end user
- Increases the productivity of the user
- Makes you a better programmer

Identify the problem?

```
SELECT * FROM SBOOK.
    CHECK: SBOOK-CARRID = 'LH' AND
                SBOOK-CONNID = '0400'.
  ENDSELECT.
```

Response Time : 452 ms

Identify the problem?

- Selection Criteria

<u>Recommended</u>

```
SELECT * FROM SBOOK
            WHERE CARRID = 'LH' AND
                        CONNID = '0400'
  ENDSELECT.
```

ResponseTime : 358 ms

<u>Explanation</u>

When a programmer gives select * even if one or two fields are to be selected, this can significantly slow the program and put unnecessary load on the entire system.

When the application server sends this request to the database server, and the database server has to pass on the entire structure for each row back to the application server. This consumes both CPU and networking resources, especially for large structures.

Thus it is advisable to select only those fields that are needed, so that the database server passes only a small amount of data back.

Identify the problem?

- Usage of Aggregate Functions

```
C4A = '000'.
SELECT * FROM T100  WHERE SPRSL = 'D' AND
                                    ARBGB = '00'.
 CHECK: T100-MSGNR > C4A.
 C4A = T100-MSGNR.
ENDSELECT.
```

Recommended

```
SELECT MAX( MSGNR ) FROM T100 INTO C4A
          WHERE SPRSL = 'D' AND
                    ARBGB = '00'.
```

Identify the problem?

- Select with view

```
SELECT * FROM DD01L
  WHERE DOMNAME LIKE 'CHAR%'
        AND AS4LOCAL = 'A'.
  SELECT SINGLE * FROM DD01T
    WHERE   DOMNAME    = DD01L-DOMNAME
        AND AS4LOCAL   = 'A'
        AND AS4VERS    = DD01L-AS4VERS
        AND DDLANGUAGE = SY-LANGU.
ENDSELECT.
```

Recommended

```
 SELECT * FROM DD01V
 WHERE DOMNAME LIKE 'CHAR%'
       AND DDLANGUAGE = SY-LANGU.
ENDSELECT.
```

Identify the problem?

- **Select … Into table**

```
REFRESH X006.
SELECT * FROM T006 INTO wa_tab.
  APPEND wa_tab.
ENDSELECT .
```

Recommended

```
SELECT * FROM T006 INTO TABLE it_tab.
```

Identify the problem?

- **Copying internal tables**

```
REFRESH TAB_DEST.
LOOP AT TAB_SRC INTO TAB_DEST.
  APPEND TAB_DEST.
ENDLOOP.
```

Recommended

```
TAB_DEST[] = TAB_SRC[].
```

Identify the problem?

- **Modifying a set of lines**

```
LOOP AT TAB.
  IF TAB-FLAG IS INITIAL.
    TAB-FLAG = 'X'.
  ENDIF.
  MODIFY TAB.
ENDLOOP.
```

<u>Recommended</u>

```
 TAB-FLAG = 'X'.
MODIFY TAB TRANSPORTING FLAG
          WHERE FLAG IS INITIAL.
```

Identify the problem?

- **Deleting a sequence of lines**

```
DO 101 TIMES.
  DELETE TAB_DEST INDEX 450.
ENDDO.
```

<u>Recommended</u>

```
DELETE TAB_DEST FROM 450 TO 550.
```

Techniques

Techniques

1. Unused/Dead code
2. Subroutine Usage
3. Usage of IF statements
4. CASE vs. nested Ifs
5. MOVE statements
6. SELECT and SELECT SINGLE
7. Small internal tables vs. complete internal tables
8. Row-level processing and SELECT SINGLE
9. READing single records of internal tables
10. SORTing internal tables
11. Number of entries in an internal table
12. Nested SELECTs versus table views
13. If nested SELECTs must be used
14. SELECT * versus SELECTing individual fields
15. Avoid unnecessary statements
16. Copying or appending inter

1. Unused/Dead code

- Avoid leaving unused code in the program. Either comment out or delete the unused situation. Use program --> check --> extended program to check for the variables, which are not used statically.

2. Subroutine Usage

For good modularization, the decision of whether or not to execute a subroutine should be made before the subroutine is called.

For example:

This is better:

```
    IF f1 NE 0.
  PERFORM sub1.
ENDIF.
FORM sub1.
      ...
    ENDFORM.
```

Than this:

```
PERFORM sub1.
FORM sub1.
      IF f1 NE 0.
        ...
      ENDIF.
ENDFORM.
```

3 Usage of IF statements

-When coding IF tests, nest the testing conditions so that the outer conditions are those which are most likely to fail. For logical expressions with AND , place the mostly likely false first and for the OR, place the mostly likely true first.

Example - nested IF's:

```
IF (least likely to be true).
  IF (less likely to be true).
   IF (most likely to be true).
       ENDIF.
      ENDIF.
     ENDIF.
```

3 Usage of IF statements(Contd..)

Example - IF...ELSEIF...ENDIF :

```
IF (most likely to be true).
ELSEIF (less likely to be true).
ELSEIF (least likely to be true).
ENDIF.
```

Example - AND:

```
IF (least likely to be true) AND
        (most likely to be true).
ENDIF.
```

Example - OR:

```
IF (most likely to be true) OR
   (least likely to be true).
```

4. CASE vs. nested Ifs

- When testing fields "equal to" something, one can use either the nested IF or the CASE statement. The CASE is better for two reasons. It is easier to read and after about five nested IFs the performance of the CASE is more efficient.

5. MOVE statements

When records a and b have the exact same structure, it is more efficient to MOVE a TO b than to MOVE-CORRESPONDING a TO b.

```
MOVE BSEG TO *BSEG.
```

is better than

```
MOVE-CORRESPONDING BSEG TO *BSEG.
```

6. SELECT and SELECT SINGLE

- When using the SELECT statement, study the key and always provide as much of the left-most part of the key as possible. If the entire key can be qualified, code a SELECT SINGLE not just a SELECT. If you are only interested in the first row or there is only one row to be returned, using SELECT SINGLE can increase performance by up to three times.

7. Small internal tables vs. complete internal tables

- In general it is better to minimize the number of fields declared in an internal table. While it may be convenient to declare an internal table using the type command, in most cases, programs will not use all fields in the SAP standard table.

For example:

```
types: begin of ty_bsid,
End of ty_tab.
```

Instead of this:

```
data:  t_mara like mara occurs 0 with header line.
```

Use this:

```
data: begin of t_mara occurs 0,
        matnr like mara-matnr,
            ...
            end of t_mara.
```

8. Row-level processing and SELECT SINGLE

- Similar to the processing of a SELECT-ENDSELECT loop, when calling multiple SELECT-SINGLE commands on a non-buffered table (check Data Dictionary -> Technical Info), you should do the following to improve performance:
  - o Use the SELECT into <itab> to buffer the necessary rows in an internal table, then
  - o sort the rows by the key fields, then
  - o use a READ TABLE WITH KEY ... BINARY SEARCH in place of the SELECT SINGLE command. Note that this only make sense when the table you are buffering is not too large (this decision must be made on a case by case basis).

9. READing single records of internal tables

- When reading a single record in an internal table, the READ TABLE WITH KEY is not a direct READ. This means that if the data is not sorted according to the key, the system must sequentially read the table. Therefore, you should:
  - o SORT the table
  - o use READ TABLE WITH KEY BINARY SEARCH for better performance.

10. SORTing internal tables

- When SORTing internal tables, specify the fields to SORTed.

SORT ITAB BY FLD1 FLD2.

is more efficient than

SORT ITAB.

11. Number of entries in an internal table

- To find out how many entries are in an internal table use DESCRIBE.

```
DESCRIBE TABLE ITAB LINES CNTLNS.   Sy-tfill
```

is more efficient than

```
LOOP AT ITAB.
  CNTLNS = CNTLNS + 1.
ENDLOOP.
```

12. Nested SELECTs versus table views

- Since releASE 4.0, OPEN SQL allows both inner and outer table joins. A nested SELECT loop may be used to accomplish the same concept. However, the performance of nested SELECT loops is very poor in comparison to a join. Hence, to improve performance by a factor of 25x and reduce network load, you should either create a view in the data dictionary then use this view to select data, or code the select using a join.

13. If nested SELECTs must be used

- As mentioned previously, performance can be dramatically improved by using views instead of nested SELECTs, however, if this is not possible, then the following example of using an internal table in a nested SELECT can also improve performance by a factor of 5x:

Use this:

```
form select_good.
 data: t_vbak like vbak occurs 0 with header line.
    data: t_vbap like vbap occurs 0 with header line.
 select * from vbak into table t_vbak up to 200 rows.
    select * from vbap
           for all entries in t_vbak
           where vbeln = t_vbak-vbeln.
      ...
    endselect.
endform.
```

13. If nested SELECTs must be used (Contd..)

Instead of this:

```
form select_bad.
 select * from vbak up to 200 rows.
  select * from vbap where vbeln = vbak-vbeln.
        ...
     endselect.
 endselect.
endform.
```

13. If nested SELECTs must be used (Contd..)

- Although using "SELECT...FOR ALL ENTRIES IN..." is generally very fast, you should be aware of the three pitfalls of using it:
- Firstly, SAP automatically removes any duplicates from the rest of the retrieved records. Therefore, if you wish to ensure that no qualifying records are discarded, the field list of the inner SELECT must be designed to ensure the retrieved records will contain no duplicates (normally, this would mean including in the list of retrieved fields all of those fields that comprise that table's primary key).
- Secondly, if you were able to code "SELECT ... FROM <database table> FOR ALL ENTRIES IN TABLE <itab>" and the internal table <itab> is empty, then all rows from <database table> will be retrieved.
- Thirdly, if the internal table supplying the selection criteria (i.e. internal table <itab> in the example "...FOR ALL ENTRIES IN TABLE <itab> ") contains a large number of entries, performance degradation may occur.

14. SELECT * versus SELECTing individual fields

- In general, use a SELECT statement specifying a list of fields instead of a SELECT * to reduce network traffic and improve performance. For tables with only a few fields the improvements may be minor, but many SAP tables contain more than 50 fields when the program needs only a few. In the latter case, the performace gains can be substantial. For example:

Use:

```
select vbeln auart vbtyp from table vbak
  into (vbak-vbeln, vbak-auart, vbak-vbtyp)
  where ...
```

Instead of using:

```
select * from vbak where ...
```

15. Avoid unnecessary statements

- There are a few cases where one command is better than two. For example:

Use:

append <tab_wa> to <tab>.

Instead of:

<tab> = <tab_wa>.

append <tab> (modify <tab>).

And also, use:

if not <tab>[] is initial.

Instead of:

describe table <tab> lines <line_counter>.

if <line_counter> > 0.

16. Copying or appending internal tables

- Use this:

<tab2>[] = <tab1>[]. (if <tab2> is empty)

Instead of this:

```
loop at <tab1>.
  append <tab1> to <tab2>.
endloop.
```

However, if <tab2> is not empty and should not be overwritten, then use:

```
append lines of <tab1> [from index1] [to index2] to <tab2>.
```

-

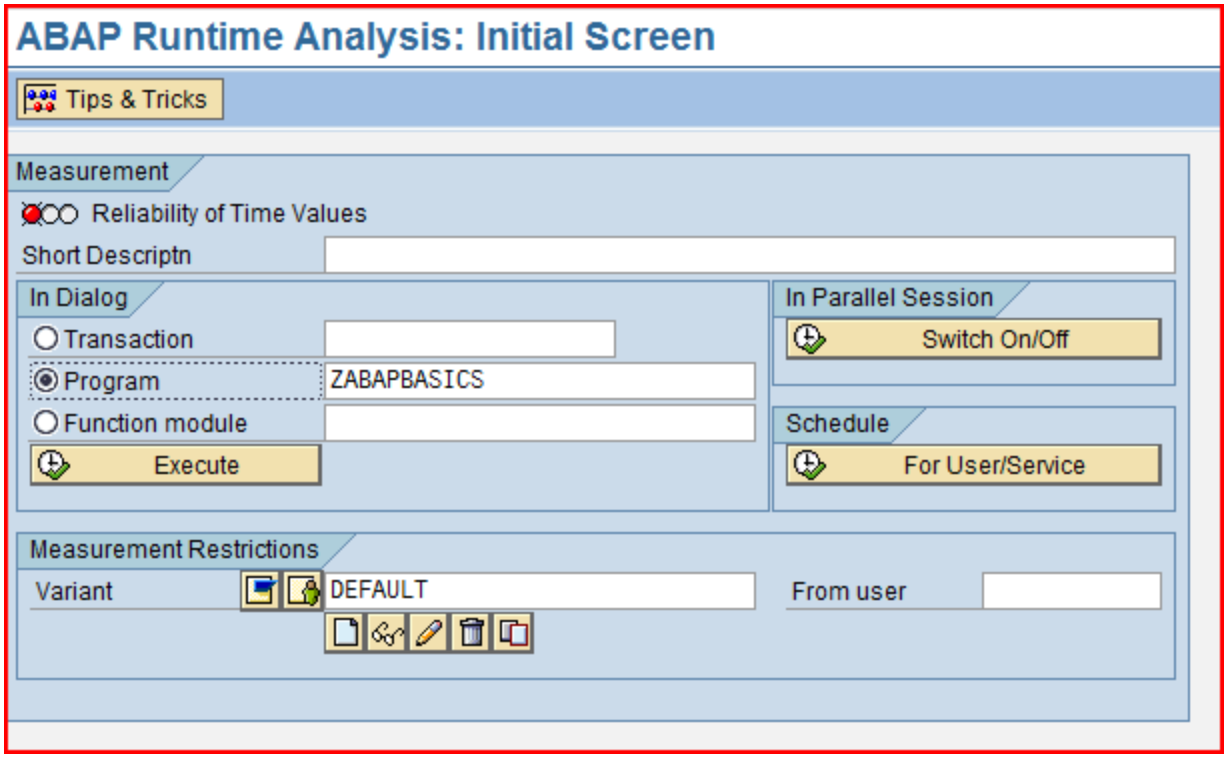
Tools

Tools

- The runtime analysis (SE30) SM30 -table TMG.
- SQL Trace (ST05)
- Code Inspector (SCI)
- Tcode: SLIN for EPC.
- 800 abaptr userecc
- ATC ABAP ON HANA

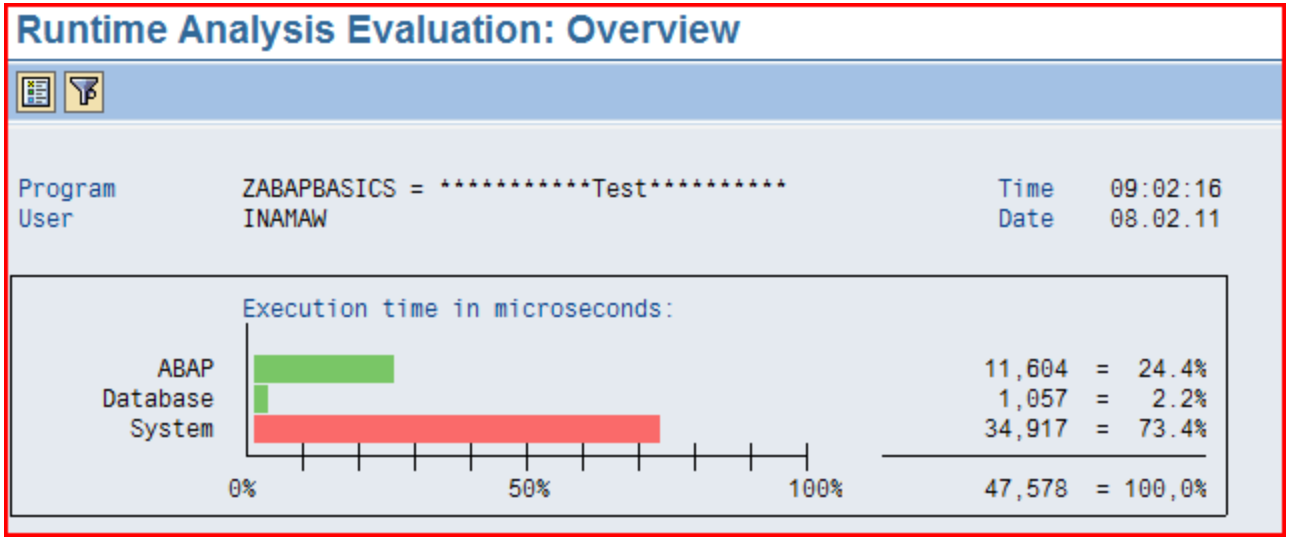
SE30 – Runtime Analysis

Initial Screen



SE30 – Runtime Analysis

Statistics Screen



SE30 – Runtime Analysis

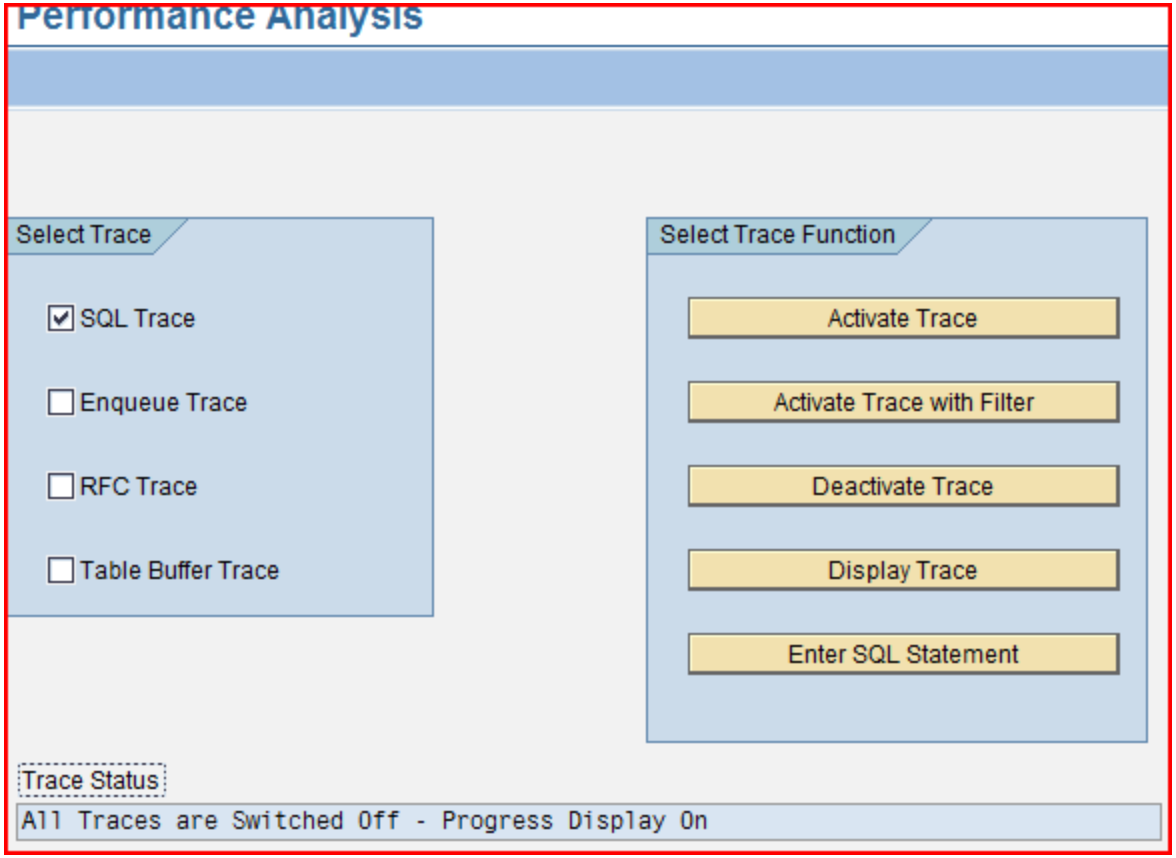
- Click on F5 to see more detailed screen as below

Runtime Analysis Evaluation: Hit List

No.	Gross	=	Net	Gross (%)	Net (%)	Call	Program Name	Type	No.	Filter
1	47,578		0	100.0	0.0	Runtime analysis		Sys.		
1	47,553		3,537	99.9	7.4	Submit Report ZABAPBASICS	SAPMS38T			
1	41,730		3,321	87.7	7.0	Program ZABAPBASICS				
1	483	=	483	1.0	1.0	Fetch T100	ZABAPBASICS	DB		OpenS
1	415	=	415	0.9	0.9	Open Cursor T100	ZABAPBASICS	DB		OpenS
1	157	=	157	0.3	0.3	Select Single TRDIR	CL_ABAP_LIST_PARSER==========CP	DB	S	OpenS
1	53	=	53	0.1	0.1	Load Report ZABAPBASICS		Sys.		
1	9	=	9	0.0	0.0	Load Report SCL_ABAP_LIST_PARSER==========CP	CL_ABAP_LIST_PARSER==========CP	Sys.		
1	2	=	2	0.0	0.0	Close Cursor T100	ZABAPBASICS	DB		OpenS
1	0	=	0	0.0	0.0	Perform not found BEFORE_EVENT	ZABAPBASICS			

ST05- SQL Trace

- Initial Screen



ST05- SQL Trace

- Analysis Screen

Trace List

DDIC information | Explain

Transaction ST05	Work process no 0	Proc. Type DIA	Client 208	User INAMAW	TransGUID 4D502CD2DBE20358E100800024530C60	Date 08.02.2011

Duration	Obj. name	Op.	Recs.	RC	Statement
15	TFDIR	REOPEN		0	SELECT WHERE "FUNCNAME" = 'SQLT_SYTIM_TO_LOTIM'
3,622	TFDIR	FETCH	1	0	

Transaction SE38	Work process no 0	Proc. Type DIA	Client 208	User INAMAW	TransGUID 4D502CDADBE20358E100800024530C60	Date 08.02.2011

Duration	Obj. name	Op.	Recs.	RC	Statement
6	PROGDIR	REOPEN		0	SELECT WHERE "NAME" = 'ZABAPBASICS' AND "STATE" = 'A'
543	PROGDIR	FETCH	1	0	
2	DWWASYNC	REOPEN		0	SELECT WHERE "UNAME" = 'INAMAW'
120	DWWASYNC	FETCH	1	0	
3	DWINACTIV	REOPEN		0	SELECT WHERE "OBJECT" = 'REPS' AND "OBJ_NAME" = 'ZABAPBASICS'
147	DWINACTIV	FETCH	0	1403	
2	DWWASYNC	REOPEN		0	SELECT WHERE "UNAME" = 'INAMAW'
108	DWWASYNC	FETCH	1	0	
2	DWINACTIV	REOPEN		0	SELECT WHERE "OBJECT" = 'REPS' AND "OBJ_NAME" = 'ZABAPBASICS'
103	DWINACTIV	FETCH	0	1403	
2	TRDIR	REOPEN		0	SELECT WHERE "NAME" = 'ZABAPBASICS'
143	TRDIR	FETCH	1	0	
4	T100	REOPEN		0	SELECT WHERE "SPRSL" = 'D' AND "ARBGB" = '00'
245	T100	FETCH	1	1403	
3	TRDIR	REOPEN		0	SELECT WHERE "NAME" = 'ZABAPBASICS'
124	TRDIR	FETCH	1	0	

Transaction ST05	Work process no 0	Proc. Type DIA	Client 208	User INAMAW	TransGUID 4D502CE8DBE20358E100800024530C60	Date 08.02.2011

Duration	Obj. name	Op.	Recs.	RC	Statement
6	TFDIR	REOPEN		0	SELECT WHERE "FUNCNAME" = 'PERFORMANCE_TRACE_OFF'
461	TFDIR	FETCH	1	0	

QUESTIONS?