

SAP application performance

SAP DEVELOPMENT

ABAP PROGRAMMING – PERFORMANCE TIPS AND TRICKS

GALLERY | MAY 11, 2014 | ERIC | [LEAVE A COMMENT](#)

When you are coding an ABAP program, you often find that the ABAP program can be coded in different ways to do the same job, you are wondering which way would give a better performance. When you are tuning the ABAP program performance, you encounter an expensive ABAP statement in SAP ST12/SE30 trace, you are wondering why it is so expensive and what are other BAP coding alternatives to replace the expensive one to improve the ABAP program performance. Tips and tricks utility built in standard SAP transaction SAT/SE30 might help you with those questions. SAP SAT/SE30 is comparing performance of common ABAP techniques/statements used by ABAP developers via ABAP performance examples. So understanding those trips and tricks can help ABAP developers to avoid common performance pitfalls during ABAP program development phase. It can also help you when you need to tune ABAP program performance. This post would cover

1. Introduction of ABAP performance tips and tricks,
2. How to access ABAP performance tips and tricks examples,
 1. Preferred coding techniques for SAP SQL interface – database table access,
 2. Preferred coding techniques for ABAP internal table access,
3. How to use performance tips and tricks examples in ABAP program development and performance tuning.

1 Introduction of SAP ABAP performance tips and tricks

SAP SE30/SAT organizes ABAP coding performance tips and tricks into following groups or categories to make it easier for you to identify specific ABAP code techniques. See Table 1 for groups and their explanation.

Table 1 – ABAP performance Tips and Tricks Group

Performance Tips/Tricks Group	Explanation
----------------------------------	-------------

SQL interface	Explain which database access technique would give better performance by comparing pairs of Database table access techniques: <u>Select+ check</u> vs <u>Select + Where</u> , <u>Select and Exit</u> vs <u>Select ... Up to 1 Rows</u> , Find , Select Array select vs single select, read without index vs read with index, individual operation and mass operation etc.
Context	Obsolete. If applicable in your version, using context is better than select since common accessed data is stored in the memory and shared across SAP transactions.
Internal Tables	Explain which internal table access technique would give better performance by comparing pairs of kills like binary read vs key read, individual table operation vs mass internal table operation etc.
Typing	Explain that parameters/field-symbols whose types are specified is more efficient than parameters whose types are not specified.
If, Case...	Explain that case statement is more efficient than IF statement
Field Conversion	Explain that we should avoid un-necessary field conversion by using the same data type.
Character / String Manipulation	Explain that using string is better than character from performance point view
ABAP Objects	Explain that call method and call subroutine has no material performance difference but calling FM is slower than Method.

If you need to read a record from big internal table, you can read it via key or binary search, then you can go to “internal table” group to which one is better to your situation.

Two groups – inefficient SQL statements and inefficient ABAP internal table operation are highlighted based on my experience. They are often culprit of bad ABAP program performance related to coding. So familiar with performance coding techniques under these two groups should have priority over other groups.

2 How to access ABAP performance Tips and Tricks

To access SAP ABAP performance tips and tricks, you need to execute SAP transaction SAT/SE30, then click on button “Tips & Tricks”.

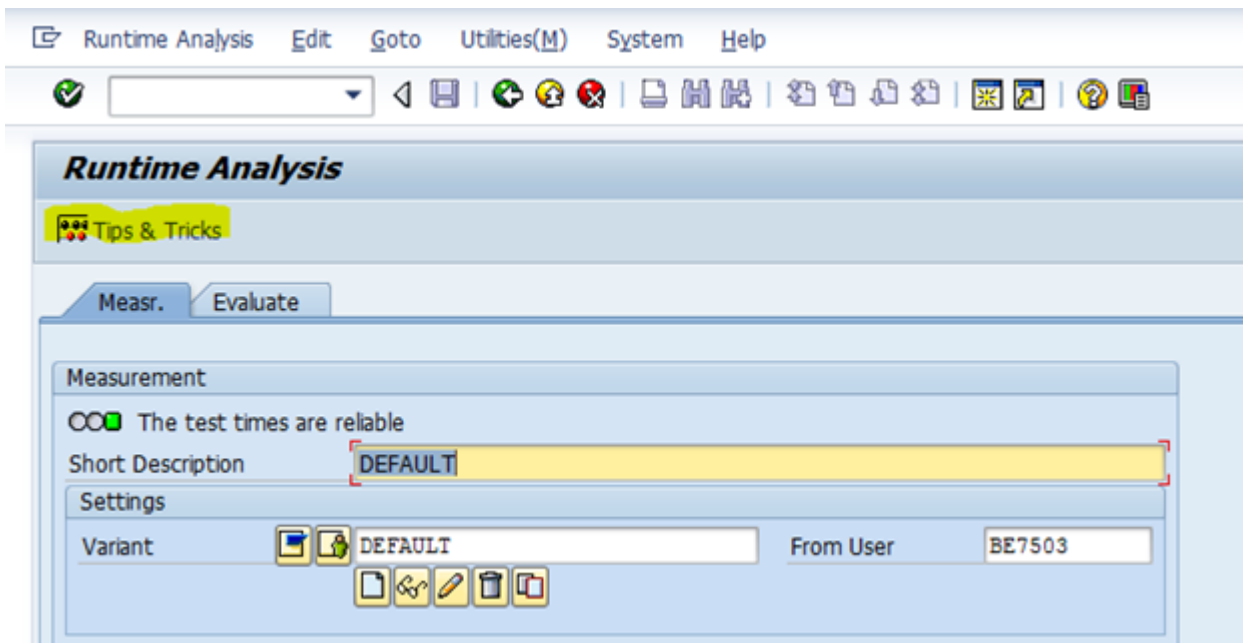


Figure 1 SE30/SAT initial screen

If you click button “Tips and Tricks” button highlighted in Figure 1, performance tips and tricks for ABAP objects screen like below Figure 2 would be displayed.

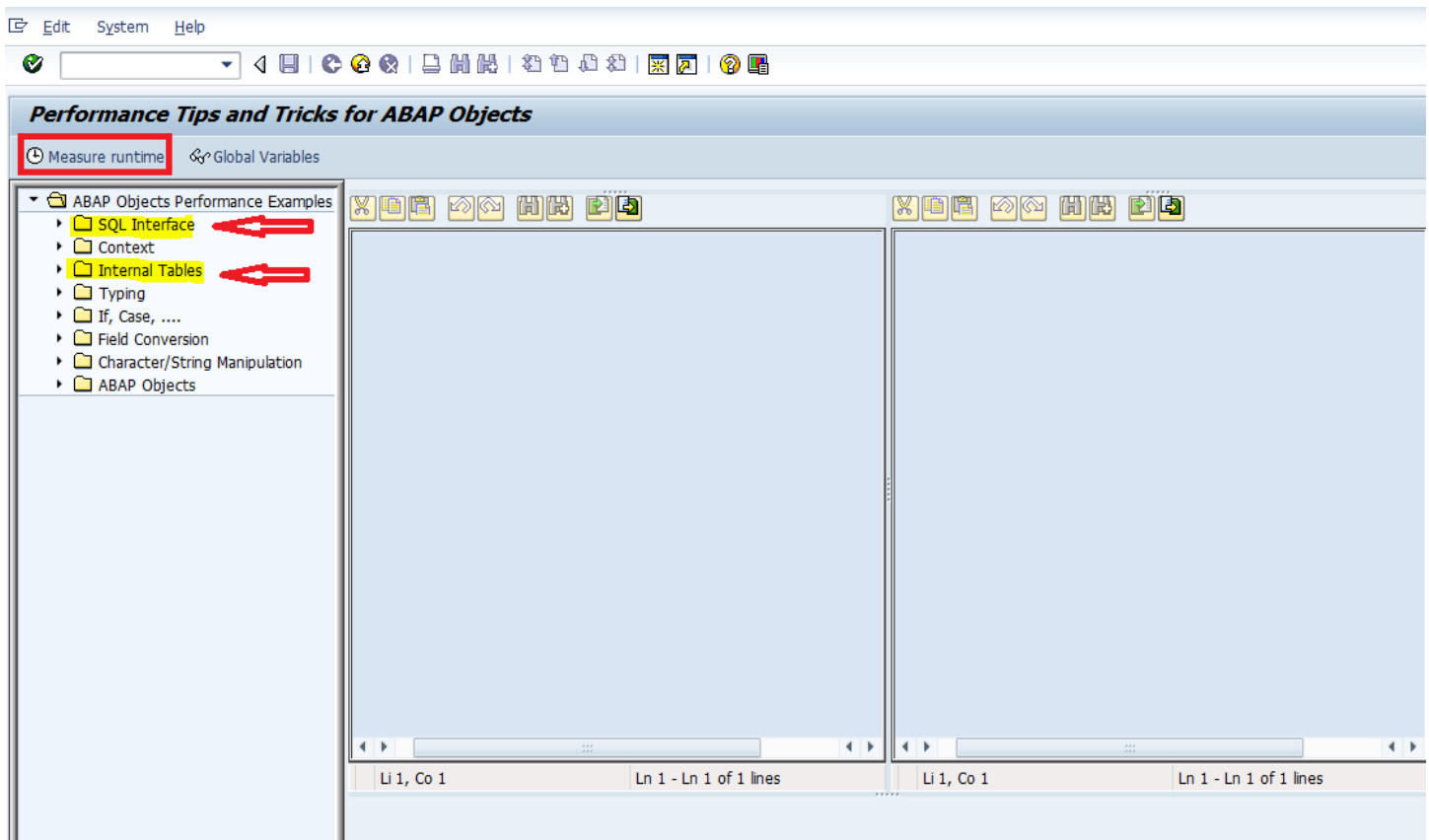


Figure 2 Se30/SAT Performance Tips and Trick Initial screen

I would cover more details related to highlighted groups “SQL interface” and “Internal table” below.

2.1 Preferred Performance coding techniques for SAP SQL interface / database table access

This group would explain preferred coding technique out of two different ways of table access. Please refer to Table 2 for details

Table 2 SQL interface – preferred SQL coding strategy

ITEM	Performance preferred	Explanation
Select + check VS. Select with where conditions	Select with where conditions	Use as much fields as possible in where clause: index might be possible with the fields and filter earlier to avoid unnecessary data traffic between DB server and application server
Select + exit Vs. Select with Up to 1 rows	Select with up to 1 rows	Select with up to 1 rows would incur less trips/traffic between DB and Application server.
Select + check VS. Select using Aggregate function	Select using Aggregate function	Avoid round trip between DB server and application server, reduce traffic
Select * Vs. Select with a list of fields	Select with a list of fields	Reduce round trip and traffic between application server and database server
Update one row VS update specified field	Update with specified fields	Reduce network traffic
Select without index VS Select with index	Select with index	Using an Index would reduce number of logical DB reads, so it would be faster than a full table scan when limited # of records are fetched.
Select without buffer support VS select with buffer support	Select with buffer	Reading buffer could be up to 100 time faster than read from db disc.
Select with Append statement VS select into table	Select into table	Reduce round trips between DB and application server
Many individual inserts VS Array insert	Array insert	Reduce round trips between DB and application server
Select + Endselect VS Select into table + loop table	Select into table + loop table	Reduce round trips between db server and application servers as well as network traffic
Nested Select Statement VS Select with view	Select with view	Reduce round trips between db server and application servers as well as network traffic

Nested Select statement VS
Select with Join

Select with join

Reduce round trips between db server and application
servers as well as network traffic

Using two Selects VS Select
using a subquery

Select with subquery

Reduce round trips between db server and application
servers as well as network traffic

In following section, I would go through some SQL performance examples.

2.1.1 Select + Check VS Select with where conditions

I clicked SQL interface group and all child groups under SQL interface, then I clicked “Select... Where” Vs. “Select + Check”, I got screen like Figure 3.

Select ... Where vs. Select + Check

Measure runtime Global Variables

ABAP Objects Performance Examples

- SQL Interface
 - Select ... Where vs. Select + Check
 - Test Existence
 - Select aggregates
 - Select with select list
 - Column Update
- Index and Buffer Support
 - Select with index support
 - Select with buffer support
- Array Operations (internal tables)
 - Select ... Into Table t
 - Array Insert VS Single-row Insert
 - Array-Select vs. Select-Endselect
- Select over more than one table
 - Select with view
 - Select with join
 - Using Subqueries
- Context
- Internal Tables
- Typing
- If, Case,
- Field Conversion
- Character/String Manipulation
- ABAP Objects

ABAP Code technique topic

Select + Check statement

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

Select with Where condition

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

Sample code

Documentation

Always specify your conditions in the Where-clause instead of checking them yourself with check-statements. The database system can then use an index (if possible) and the network load is considerably less.

Explanation

Figure 3 ABAP performance example – SELECT + Check Vs Select + Where clause

The display is divided into 3 portions in Figure 3, left panel – navigation to different group and ABAP code performance techniques, Right upper window is showing two sample ABAP codes using two code solutions which achieve same result. Right bottom window explains what is the performance-preferred ABAP coding technique.

Based on explanation in Figure 3, it is clear that “select with where conditions” has better performance than “select + check”. In another word, the sample code at right in figure 3 should have better

performance than ABAP sample code in the right. Is that true? You can actually test this to get a feeling by clicking “Measure Runtime” button in Figure 3. Upon clicking it, I got Figure 4 screen.

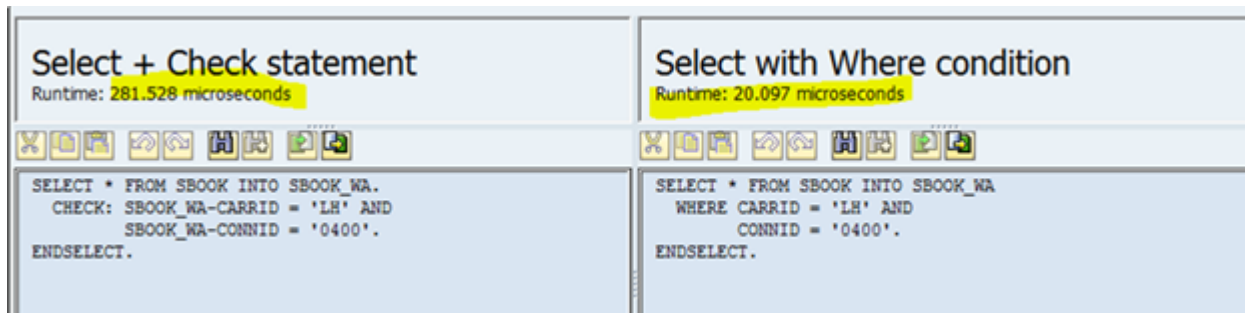


Figure 4 ABAP performance example- Select + check vs select with where conditions

In this particular case, time spent by “Select + check” example code is 10 times longer than time spent by “select with where clause” example code.

In the same fashion, you can navigate to other SQL coding techniques to get a feeling of performance difference created by different ABAP coding techniques via sample codes.

2.1.2 Select without buffer support VS Select with buffer support

Figure 5 shows that Reading data from application buffer is 10 times faster than reading data from database system.

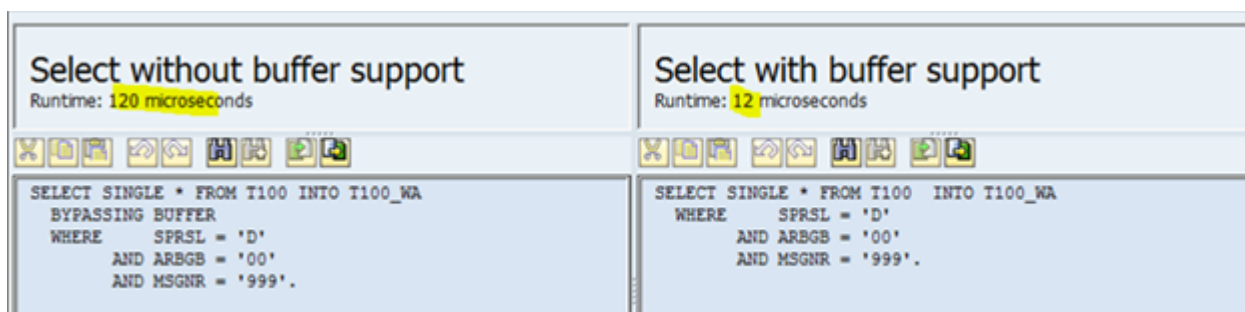


Figure 5 ABAP performance example – Select without buffer support vs Select with buffer support

2.1.3 Select into table + loop at table VS. Select Endselect

Select into table is much more efficient than Select ... Endselect. Select into table needs more “memory” to hold the result set. In normal situation, this is not a concern. When memory is a concern, you can divide the result set into smaller sets.

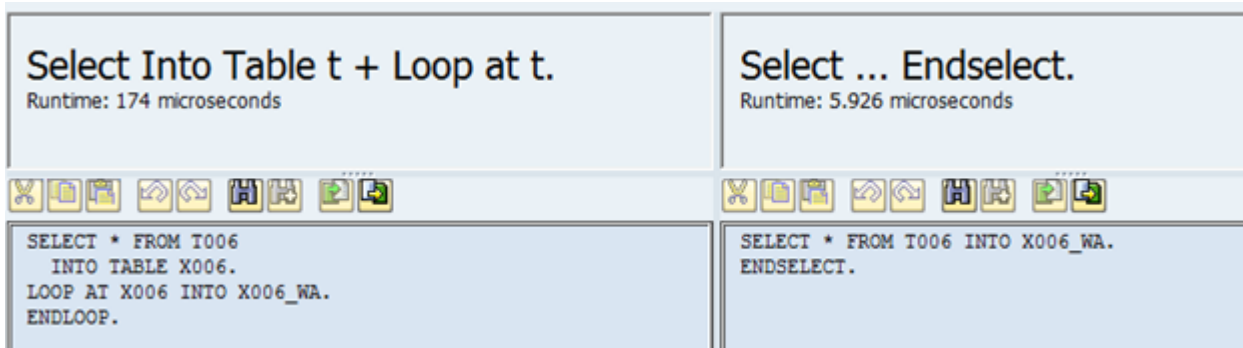


Figure 6 ABAP performance example – Select Into table + loop at table VS Select.. ENDSelect.

2.1.4 Array/Mass Insert VS single-row Insert

Array/Mass Insert is always better from performance point view since it can reduce unnecessary round trips between database and application server by utilizing full capacity of data communication bus between database and application server. It can also allow database



Figure 7 ABAP performance example – Single-line insert VS Array Insert

2.1.5 Nested selects VS Select with view

Select with view always performs better than “Nested Select”.

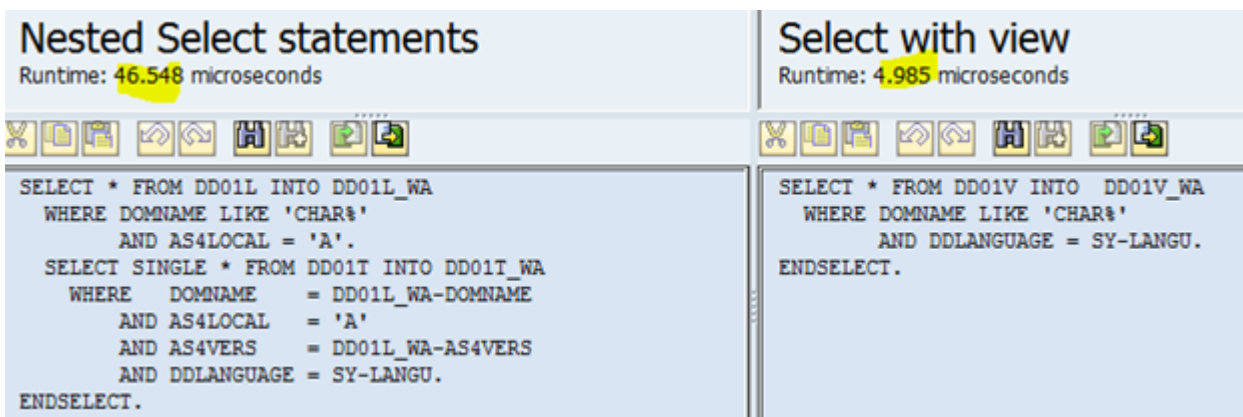


Figure 8 ABAP performance example – Nested Select VS Select with View

2.1.6 Nested select VS Select with Join

Select with join always performances better than Nested selected.



Nested Select statements Runtime: 3.348 microseconds	Select with join Runtime: 1.584 microseconds
 <pre> SELECT * FROM SPFLI INTO SPFLI_WA. SELECT * FROM SFLIGHT INTO SFLIGHT_WA WHERE CARRID = SPFLI_WA-CARRID AND CONNID = SPFLI_WA-CONNID. ENDSELECT. ENDSELECT. </pre>	 <pre> SELECT * INTO WA FROM SPFLI AS P INNER JOIN SFLIGHT AS F ON P-CARRID = F-CARRID AND P-CONNID = F-CONNID. ENDSELECT. </pre>

Figure 9 ABAP performance example – Nested Select Vs Select join

2.2 Preferred Performance coding techniques for ABAP internal table operation

Please refer to Table-3 for some ABAP code techniques which are important to application performance. They are selected from what offered by SAP SAT/SE30 tips and tricks based on my performance experience.

Table 3 SAT/SE30 ABAP internal table handling performance tips and tricks

ITEM	Performance preferred	Explanation
Linear search VS Binary search	Binary search	If internal tables are assumed to have many (>20) entries, a linear search through all entries is very time-consuming. Try to keep the table ordered and use binary search or used a table of type SORTED TABLE. If TAB has n entries, linear search runs in O(n) time, whereas binary search takes only O(log2(n)).
Secondary index Vs No Secondary	Using secondary index for repeated accesses	When an internal table is accessed with different keys repeatedly, create your own secondary index. With a secondary index, you can replace a linear search with a binary search plus an index access.
LOOP internal table and CHECK VS LOOP ... WHERE	Loop... where	LOOP ... WHERE is faster than LOOP/CHECK because LOOP ... WHERE evaluates the specified condition internally. As with any logical expressions, the performance is better if the operands of a comparison share a common type. The performance can be further enhanced if LOOP ... WHERE is combined with FROM i1 and/or TO i2, if possible.
Sorted table VS Hashed Table	Sorted table for repeated accesses.	Hashed tables are optimized for single entry access. The entries have no specific order. Therefore, a partial sequential access cannot be optimized. Every entry must be checked to match the condition (full

Hashed table for
single access

table scan). Sorted tables are ordered ascendingly by their key components. If a partial key of the form "k1 = v1 AND ... AND kn = vn" where k1 .. kn matches a left part of the table key, the access is optimized by the kernel. In that case, only the matching entries are visited.

Copying a table via loop
VS equal

Equal operation
better

Internal tables can be copied by MOVE just like any other data object. If an internal table itab has a header line, the table itself is accessed by itab[.].

Join table – Loop table 1
then read table 2 VS loop
table1 then read table 2
with index

Loop table 1 then read
2nd table with an
index

If ITAB1 has n1 entries and ITAB2 has n2 entries, the time needed for joining ITAB1 and ITAB2 with the straightforward algorithm is $O(n1 * \log_2(n2))$, whereas the parallel cursor approach takes only $O(n1 + n2)$ time. The above parallel cursor algorithm assumes that ITAB2 is a secondary table containing only entries also contained in primary table ITAB1. If this assumption does not hold, the parallel cursor algorithm gets slightly more complicated, but its performance characteristics remain the same.

In following sections, performance code examples from SAT/SE30 are covered for selected topics.

2.2.1 Read internal table via key VS Read Internal table via binary search

Figure 10 show It is more efficient to read an internal table via binary search if the table entries is huge. This benefit might be more significant if you need to read the table many times.

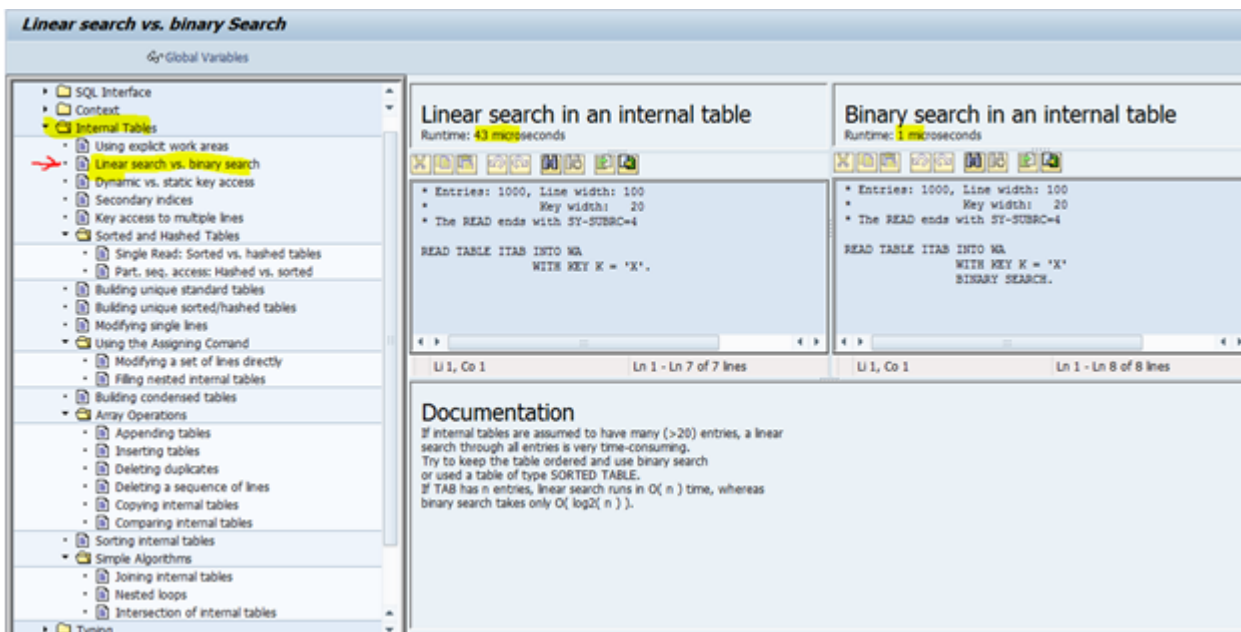


Figure 10 ABAP performance example – Linear against Binary search in an internal table

2.2.2 Read internal table without secondary index VS secondary index

Reading an internal table using a secondary index could be more efficient than read via a key.

The screenshot shows a performance comparison tool titled "Creating a secondary index to avoid linear search". It compares two methods of reading an internal table: linear search and binary search using a secondary index.

Method	Runtime	Code Snippet
No secondary index => linear search	Runtime: 24 microseconds	<pre> * Entries: 1000, Line width: 100 * Key width: 20 * The READ locates the 500th entry. READ TABLE ITAB INTO WA WITH KEY DATE = SY-DATUM. IF SY-SUBRC = 0. ... ENDIF.</pre>
Binary search using secondary index	Runtime: 3 microseconds	<pre> * Entries: 1000, Line width: 100 * Key width: 20 * The READ locates the 500th entry. READ TABLE SEC_IDX INTO SEC_IDX_WA WITH KEY DATE = SY-DATUM BINARY SEARCH. IF SY-SUBRC = 0. READ TABLE ITAB INTO WA INDEX SEC_IDX_WA-INDEX. ... ENDIF.</pre>

Figure 11 ABAP performance example – Read internal table using index VS no-index

2.2.3 Read internal table via combination of Loop and Check VS read internal table via Loop... Where

Reading an internal table with loop... where statements is more efficient than loop + check statements.

The screenshot shows a performance comparison tool titled "Key access with LOOP/CHECK" vs "Key access with LOOP ... WHERE". It compares two methods of reading an internal table: using a loop with a check statement vs using a loop with a where statement.

Method	Runtime	Code Snippet
Key access with LOOP/CHECK	Runtime: 18 microseconds	<pre> * Entries: 1000, Line width: 500 * Key width: 20 * 5 entries of which match the key condition LOOP AT ITAB INTO WA. CHECK WA-K = 'X'. ... ENDLOOP.</pre>
Key access with LOOP ... WHERE	Runtime: 8 microseconds	<pre> * Entries: 1000, Line width: 500 * Key width: 20 * 5 entries of which match the key condition LOOP AT ITAB INTO WA WHERE K = 'X'. ... ENDLOOP.</pre>

Figure 12 ABAP performance example – Loop and Check VS Loop where

2.2.4 Read internal table – Hashed table VS sorted table

When you need to read the internal table many times, using sorted table is more efficient than using hashed table.

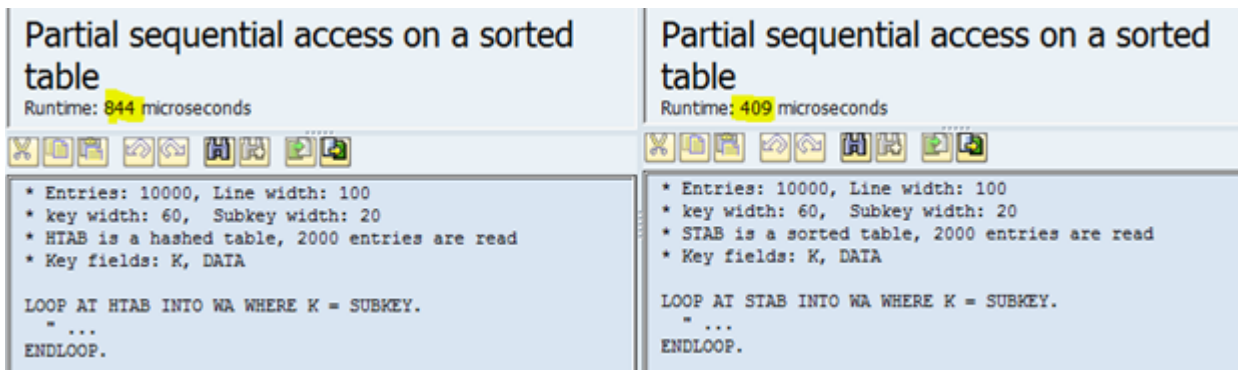


Figure 13 ABAP performance example – Sorted table VS Hashed table

2.2.5 Copy internal table – Loop + append and copy VS “=”

Using “=” to copy table is more efficient than using Loop + append to copy an internal table.

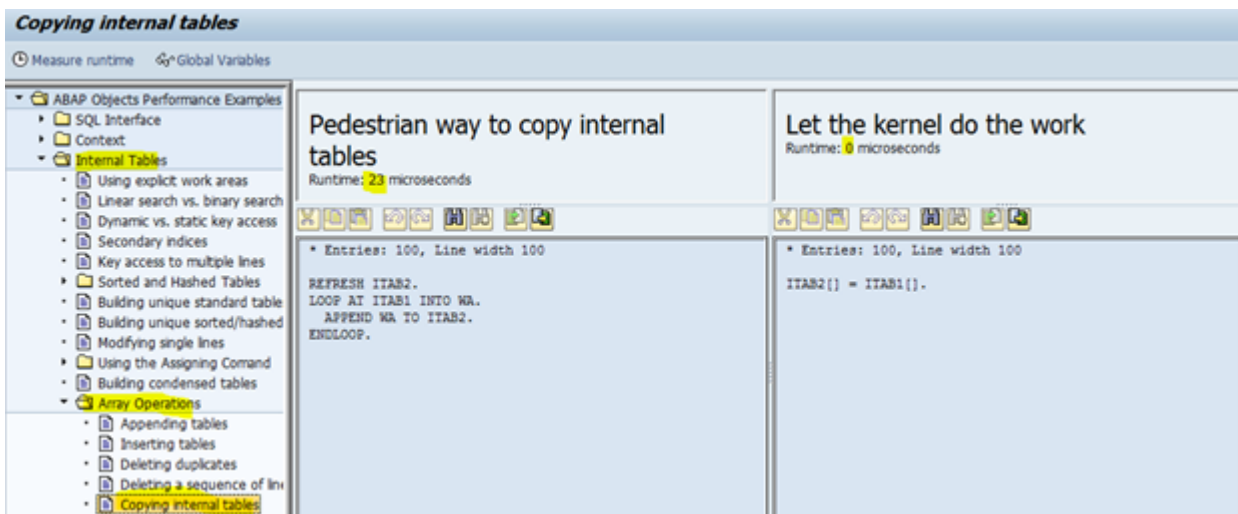


Figure 14 ABAP performance example – copy internal table

2.2.6 Join internal tables – Loop table1 and read table2 against loop table 1 and read table2 via Index

Using method of looping table 1 than reading table 2 via INDEX is more efficient than method of looping table 1 then reading table 2 without index.

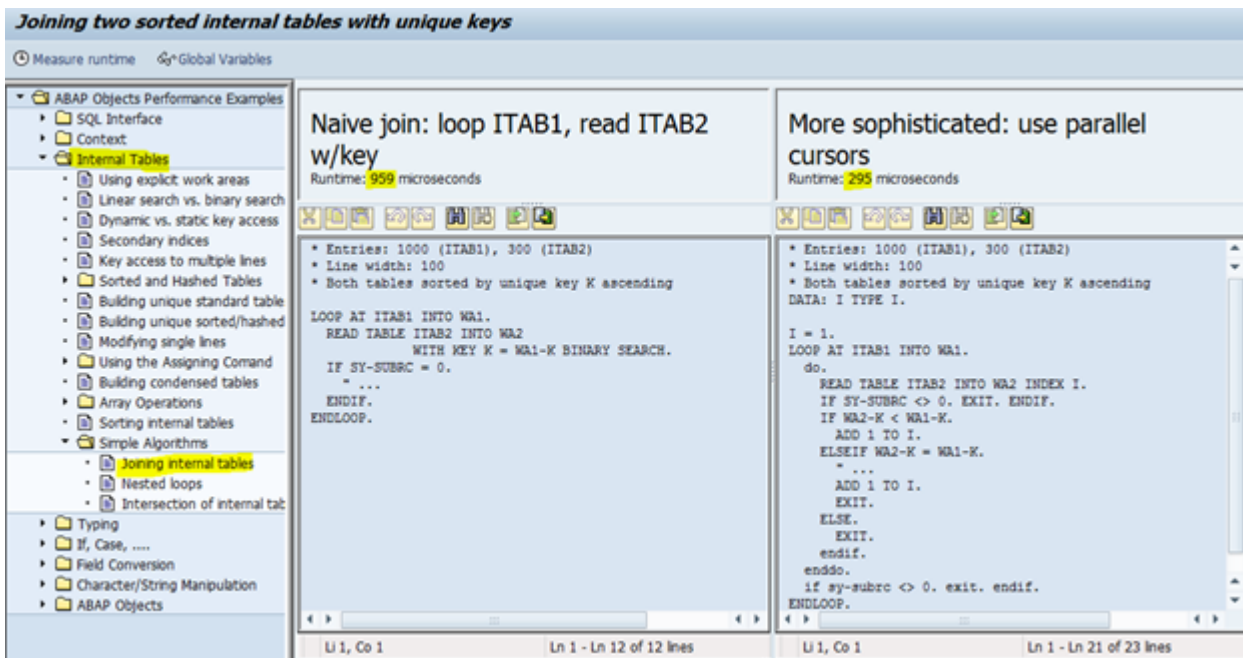


Figure 15 ABAP performance example – join internal tables

3 Testing performance of your own ABAP codes

I have seen ABAP developer to compare performance of two different sets of ABAP code which achieve the same purpose via SE30. It looks like that this can be done via SAT/SE30 menu path EDIT-> Editing -> LEFT-hand /Right-Hand as showed in Figure 16. I do not have authorization to do this in the system I owned.. Readers like you can explore this more should you be interested in it.

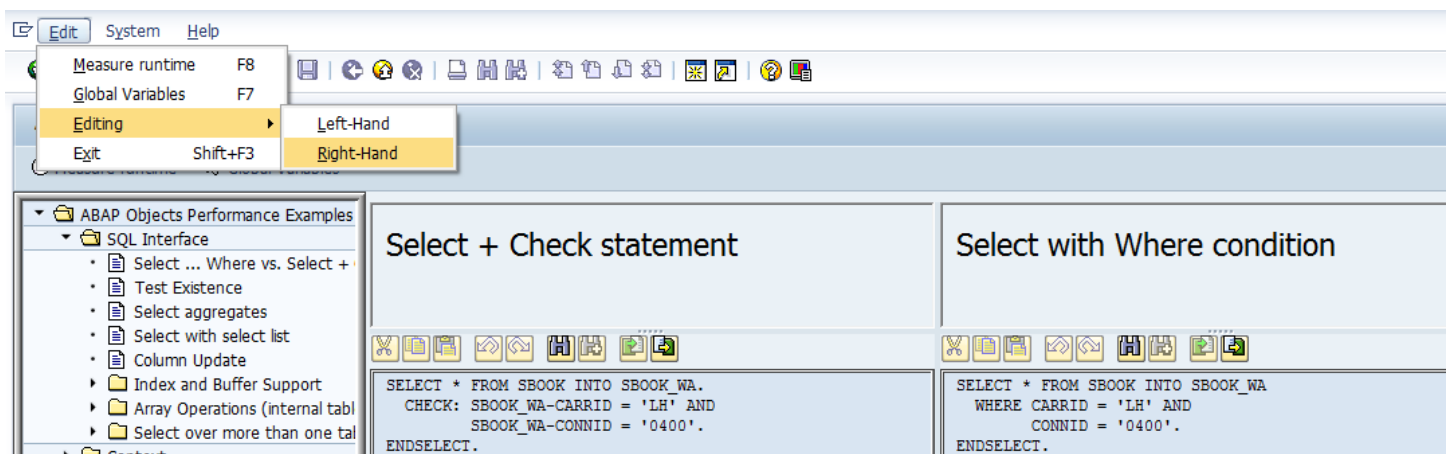


Figure 16 SAT/SE30 – your own sample codes

4 Further Information

A SAP ABAP program performance can have many contributing factors like solution design, functional configuration and program code from application performance point view. An application performance

issue can be rooted from database performance, system capacity, network, IO design as well. SAP performance tips and tricks cannot help to improve ABAP program performance issue which is rooted from solution design, functional configuration, database performance issue and etc. It can help to verify whether a program performance issue is rooted from programming techniques. In my experience, over 50% of ABAP program performance issues are related to program code. So it is important for a SAP developer to become familiar with those performance examples and performance-preferred ABAP coding techniques.

Again, not all performance examples from SAP SAT/SE30 are covered here. My intention is to cover most common/critical examples based on my experiences. You can run SAP SAT/SE30 online to review the remaining ABAP performance examples. Also, SAT/SE30 covers “common” performances examples. There are many ways to put ABAP statement together for the same purpose. It is not the expectation that SAT/SE30 provides a examples for each situation.

SE30 allows you to do ABAP runtime analysis but it would not provide much data to support SQL performance. My preferred tool to do ABAP program runtime analysis is SAP transaction ST12 which I find efficient to do performance analysis on ABAP logic operation and database access. I might come out a post on how to use SE30 to do ABAP runtime analysis in the future.

I have following posts related to SAP program performance: [how to improve a SAP ABAP program performance](#); [how to run SAP ST12 to do a performance trace](#); [how to analyze ABAP trace](#); [how to analyze SQL trace](#);