ABAP Performance and Tuning

- What tools can be used to help with performance tuning?
- What are the steps to optimise the ABAP Code?
- What is the difference between SELECT SINGLE and SELECT ... UP TO 1 ROWS?
- Which is the better - JOINS or SELECT... FOR ALL ENTRIES...?
- Does SAP publish guides and cookbooks on performance monitoring and testing?

#Avoid use of nested loops

**What tools can be used to help with performance tuning?**

ST05 is the performance trace. It contains the SQL Trace plus RFC, enqueue and buffer trace. Mainly the SQL trace is used to measure the performance of the select statements of the program.

SE30 is the Runtime Analysis transaction and can be used to measure the application performance.

SAT transaction is the replacement of the pretty outdated SE30. Provides the same functionality as SE30 plus some additional features.

ST12 transaction (part of ST-A/PI software component) is a combination of ST05 and SAT. Very powerful performance analysis tool used primarily by SAP Support.

One of the best tools for static performance analyzing is Code Inspector (SCI). There are many options for finding common mistakes and possible performance bottlenecks.

**What are the steps to optimize the ABAP Code?**

1. **DATABASE**
   1. Use WHERE clause in your SELECT statement to restrict the volume of data retrieved. Very important!!
   2. Design your Query to Use as much index fields as possible in your WHERE statement
   3. Use INNER (or OUTER under some circumstances) JOIN in your SELECT statement to retrieve the matching records at one shot
   4. Avoid using nested SELECT statements and SELECT within LOOPS, better use JOINS or FOR ALL ENTRIES. Use FOR ALL ENTRIES when the internal table is already there or the end of some processing. Try JOINS if the SELECT are right behind each other
   5. Avoid using INTO CORRESPONDING FIELDS OF TABLE during buffered access. Otherwise use the most appropriate for the program.
   6. Avoid using SELECT * and Select only the required fields from the table.
   7. Avoid using ORDER BY in SELECT statements if it differs from used index. (instead, sort the resulting internal table), because this may add additional work to the database system which is unique, while there may be many ABAP servers
   8. INDEX: Creation of Index for improving performance should not be taken without thought. Index speeds up the performance but at the same time adds two overheads namely; memory and insert/append performance. When INDEX is created, memory is used up for storing the index and index sizes can be quite big on large transaction tables! When inserting new entry in the table, all the index's are updated. More index more time. More the amount of data, bigger the indices, larger the time for updating all the indices
   9. Avoid Executing an identical Select (same SELECT, same parameter) multiple times in the program. Buffer in your abap code.
   10. Avoid using join statements if adequate standard views exist no performance impact

2. **TABLE BUFFER:**
   1. Defining a table as buffered (SE11) can help in improving the performance but this has to be used with caution. Buffering of tables leads to data being read from the buffer rather than from table. Buffer sync with table happens periodically, only if something changes which is happen rarely. If this table is a transaction table chances are that the data is changing for a particular selection criteria, therefore application tables are usually not suited for table buffering. Using table buffering in such cases is not recommended. Use Table Buffering for configuration data and sometimes for Master Data...
   2. Avoid using complex Selects on buffered tables-, because SAP may not be able to interpret this request, and may transmit the request to the database- The code inspector tells which commands bypass the buffer

3. **Internal tables**
   1. Use HASHED tables where-ever possible. Otherwise SORTED tables. STANDARD tables should be the last choice.
   2. Use assign instead of into in LOOPs for table types with large work areas, if the data is being modified.
   3. When in doubt call transaction SE30 and check your code.
   4. If you must use a STANDARD table and you are using a READ, sort the table appropriately and use the addition BINARY SEARCH to speed up the search.

4. **Miscellaneous**
   1. PERFORM : When writing a subroutine, always provide type for all the parameters. This reduces the overhead which is present when system determines on it's own each type from the formal parameters that are passed. It also makes for more robust programming.

back to top
What is the difference between SELECT SINGLE and SELECT ... UP TO 1 ROWS?

- **SELECT SINGLE** and **SELECT UP TO n ROWS** return the first matching row/rows for the given condition. It may not be unique, if there are more matching rows for the given condition.
- With ORACLE database system, **SELECT SINGLE** is converted into **SELECT ... UP TO 1 ROWS**, thus they are exactly the same in that case. The only difference is the ABAP syntax prevents from using ORDER BY with **SELECT SINGLE**, but it is allowed with **SELECT ... UP TO 1 ROWS**. Thus, if several records may be returned and we want to get the highest record for example, **SELECT SINGLE** cannot be used, but **SELECT ... UP TO 1 ROWS WHERE ... ORDER BY ...** may be used.

Which is the better - JOINS or SELECT... FOR ALL ENTRIES...?

In most scenarios INNER JOIN performs better than FOR ALL ENTRIES, and should be used first. Only if there are performance issues should FOR ALL ENTRIES be considered, and careful measurements taken before and after to validate whether there really are performance gains.

The effect of FOR ALL ENTRIES needs to be observed first by running a test program and analyzing SQL trace. Certain options set by BASIS can cause FOR ALL ENTRIES to execute as an ‘OR’ condition. This means if the table being used FOR ALL ENTRIES has 3 records, SQL Trace will show 3 SQL's getting executed. In such a case using FOR ALL ENTRIES is useless. However of the SQL Trace shows 1 SQL statement it's beneficial since in this case FOR ALL ENTRIES is actually getting executed as an IN List.

JOINS are recommended over FOR ALL ENTRIES. There is no real limit to the number of tables that can be joined; however greater complexity can make maintenance harder, and if there are problems with the join, make it harder to resolve them. If the JOIN is being made on fields which are key fields in both the tables, it reduced program overhead and increases performance.

In some scenarios, you are presented with an internal table. In these situations, you may have no choice but to use FOR ALL ENTRIES.

Here is a code with join:

```plaintext
SELECT A~VBELN A~KUNNR A~KUNAG B~NAME1
INTO TABLE I_LIKP
FROM LIKP AS A
   INNER JOIN KNA1 AS B
     ON A~KUNNR = B~KUNNR.
* For with limited data using for all entries:
* Minimize entries in I_likp by deleting duplicate kunnr.
LOOP AT I_LIKP INTO W_LIKP.
  W_LIKP2-KUNAG = W_LIKP-KUNAG.
APPEND W_LIKP2 TO I_LIKP2.
ENDLOOP.
SORT I_LIKP2 BY KUNNR.
DELETE ADJACENT DUPLICATES FROM I_LIKP2 COMPARING KUNNR.
* GET DATA FROM kna1
IF NOT I_LIKP2[] IS INITIAL.
SELECT KUNNR NAME1
INTO TABLE I_KNA1
FROM KNA1
FOR ALL ENTRIES IN I_LIKP2
   WHERE KUNNR = I_LIKP2-KUNNR.
ENDIF.
```

User Collect Statement to do Sum in the internal table.

Instead of using logic to do summation use collect statement. COLLECT is especially efficient with HASHED tables.

Avoid use of nested loops

For example: if there is a loop like this. **Condition added, otherwise there is no optimization:**

```plaintext
LOOP AT ITAB1.
  LOOP AT ITAB2 WHERE F1 = ITAB1-F1.
.....
ENDLOOP.
END LOOP.
```
in the production environment it may be possible that such a loop takes a lot of time and dumps.

Instead we can use ... **BINARY SEARCH added, otherwise no improvement!!! Better still - use a HASHED or SORT TABLE.**

Error rendering macro 'code': Invalid value specified for parameter 'lang'

```abap
SORT ITAB2 BY F1.
LOOP AT ITAB1.
  READ TABLE ITAB2 WITH KEY F1 = ITAB1- BINARY SEARCH. "f1 is any field of itab1
  IF SY-SUBRC = 0.
    IDX = SY-TABIX.
    LOOP AT ITAB2 FROM IDX.
      IF ITAB2-F1 <> ITAB1-F1.
        EXIT.
      ENDIF.
    ....
  ENDLOOP.
ENDIF.
ENDLOOP.

If you have a sorted table - the internal table can be read like this:

Error rendering macro 'code': Invalid value specified for parameter 'lang'

```abap
TYPES: BEGIN OF ITAB,
  F1 TYPE MARA-MATNR,
  ....
  *NOT ONLY THE KEYFIELD !!
END OF ITAB.
DATA: ITAB2 TYPE SORTED TABLE OF ITAB WITH UNIQUE KEY F1.
LOOP AT ITAB1.
  LOOP AT ITAB2 WHERE F1 = ITAB1. "f1 is any field of itab1
  ....
  ENDLOOP.
ENDLOOP.
```