

# The Power of Performance Optimized ABAP and Parallel Cursor in SAP BI



## Applies to:

SAP BI 7.0, SAP ABAP. For more information, visit the [Business Intelligence homepage](#).

## Summary

The objective of the article is to outline & explain the performance considerations while using ABAP and parallel cursor technique in context to SAP BI. The document will also showcase the extent of performance improvement that can be derived using the technique mentioned above.

**Author:** Gundeep Singh, Deepak Somani

**Company:** Accenture

**Created on:** 11 February 2010

## Author Bio



Gundeep Singh is working as SAP BI Consultant in Accenture Services Private Ltd and having extensive experience in implementation of BI/ABAP projects specializing in SCM areas.



Deepak Somani is working as SAP BI Consultant in Accenture Services Private Ltd and having extensive experience in implementation of BI/ABAP projects across various functional areas.

## Table of Contents

Pre-Requisites .....	3
Database Access Using Select .....	3
Internal Table Operations .....	4
Parallel Cursor Technique: Overview .....	5
Need of Nested loop .....	5
Example/Illustration .....	5
Example of a Nested Loop .....	6
Parallel Cursor Approach: Demystified .....	6
Sample Code & Demonstration .....	6
Nested Loop Using Parallel Cursor: .....	6
Performance Data Comparison .....	7
Related Content .....	7
Disclaimer and Liability Notice .....	8

## Pre-Requisites

- Basic understanding of SAP ABAP programming language
- Basic understanding of dataflow in SAP BI 7.0.

Efficient Programming involves solving a problem / defining logic as fast as possible while using system resources as sparingly as possible. In context to SAP BI, when multimillion records form the crux of the tool, the significance of efficient ABAP codes is very high. It is highly recommended to write performance optimized & fine tuned codes for overall health of the system. As a part of this write up we will look at some of the very basic and simple to use techniques that if followed and adhered to can result in highly optimized BI systems.

## Database Access Using Select

Some best practices and common tips for considering performance while writing Select statements:

### Use Where Clause



```
select field1 from dbtab1 into var1.
endselect.
```



```
select field1 field2 from dbtab1 into var1 var2
where field 1 = X.
endselect.
```

### Use Select Single



```
Select * from
dbtab1 where fld2 = X.
endselect.
if sy-subrc <> 0.
endif.
```



```
Select single * from
dbtab1 where fld2 = X.

if sy-subrc <> 0.
endif.
```

### Avoid Select \*



```
Select * from
dbtab1 where fld2 = X.
endselect.
if sy-subrc <> 0.
endif.
```



```
Select fld1 fld2 fld3 from
dbtab1 into var1 var2 var3
where fld2 = X.
endselect.
```

## Use 'For All Entries'

```

Loop at source_package.
  Select single F1 F2 F3 into wa_xx from YYY
  where F1 = source_package-F1.
Endloop.

```



```

DATA : nb type n.
Describe table it_XX lines nb.
If nb is not initial.
  Select F1 F2 F3 from YYY into it_XX for all entries in source_package
  Where F1 = source_package-F1.
Endif.

```



Making use of For All Entries as shown above ensures that we are selecting only relevant records maintaining best performance

## Internal Table Operations

### Deleting a Record – for single value condition

```

Loop at source_package assigning <source_fields>.
  If <source_fields>-F1 = 'X'.
    Delete source_package.
  Endif.
Endloop.

```



```
Delete source_package where F1 = 'X'.|
```



### Deleting a Record – for multiple value condition

```

DATA : var1 type F1. "i.e. the field for which we are creating SELECT-OPTIONS
SELECT-OPTIONS : s_g1 for Var1.
DATA : wa like line of s_g1.
wa-sign = 'I'.
wa-option = 'EQ'.
wa-low = 'ABCD'.
append wa to s_g1.
wa-sign = 'I'.
wa-option = 'EQ'.
wa-low = 'EFGH'.
append wa to s_g1.
Delete source_package where F1 in s_f1.

```

## Modifying a Record

```
Loop at source_package into wa_xx.
Wa_xx-F1 = 'X'.
Modify source_package.
Endloop.
```



```
FIELD-SYMBOLS: <SOURCE_FIELDS> TYPE _ty_s_sc_1.
Loop at source_package assigning <source_fields>
<source_fields>-F1 = 'X'.
EndLoop.]
```



Modifying a record of an internal table consumes time as it copies the entire dataset into a separate work area. Instead, making use of Field Symbols improves the performance by 10 times.

## Copying a Record/Internal Tables

```
Loop at itab1 into wa1.
Clear wa2.
Move-corresponding wa1 to wa2.
Append wa2 to itab2.
Endloop.
```



```
itab1[] = itab2[].
```



To Copy data from one internal table into another ( where the two internal tables are exactly the same ) , make use of the statement as shown above.

## Parallel Cursor Technique: Overview

### Need of Nested loop

IN SAP BI routines/extractors we come across a lot of situations wherein we loop a header internal table and carry out some processing for each of the lines in the corresponding Item level internal table. To perform the same, we end up using nested loops which have a great impact on the performance and result in long execution times. To overcome the problem mentioned above, we can make use of parallel cursor technique/approach wherein we maintain a cursor which holds the value of the index from where the subsequent search should start. This will result in vastly optimized ABAP components which would further facilitate faster data loading at every step of BI data flow.

### Example/Illustration

Say our inner internal table has 1000 records. The second loop i.e the inner loop will operate on the basis of where condition as per the first loop. To suffice this where condition, the control will actually search the internal table starting from the first record itself, moving till the last (1000th record). The time it takes for finding the record will increase for the records that have matches at the end of the internal table. That means the search will take more time to find the 1000th record than the time it takes to search for the 1st or 2nd record. This as a result increases the loop & run time which further increases the data load time especially in a real time scenario when the data count is huge and enormous.

## Example of a Nested Loop

Suppose we have two internal tables IT\_EKKO and IT\_EKPO containing purchase order header and item level data and we need to extract some item level details on the basis of PO number which is common key in both the tables. We will have to put a nested loop here.

Suppose there are 1000 records in table IT\_EKPO and for PO 123 there are three records available in IT\_EKPO table.

```
DATA : it_ekko TYPE STANDARD TABLE OF ekko,
      wa_ekko TYPE ekko,
      it_ekpo TYPE STANDARD TABLE OF ekpo,
      wa_ekpo TYPE ekpo.
DATA : lv_tabix TYPE sy-tabix.
SELECT * FROM ekko INTO TABLE it_ekko.
SELECT * FROM ekpo INTO TABLE it_ekpo.
LOOP AT it_ekko INTO wa_ekko.
  LOOP AT it_ekpo INTO wa_ekpo WHERE ebeln = wa_ekko-ebeln.
    Logical ABAP statements .....
  ENDLLOOP.
ENDLOOP.
```

Here when WA\_EKKO-EBELN =123 the inner loop will execute 1000 times and statement inside the loop will execute 3 times. This will result in higher data load runs and poor performance. We can optimize the above using parallel cursor technique as elaborated below:

## Parallel Cursor Approach: Demystified

Parallel cursor is the technique to increase the performance of the program, when there are nested loops. The approach uses a variable using which we can maintain a cursor which will avoid the search to start all over again, from the first record. Instead it will control the loop to start the search from a specified index. This as a result makes the search faster and thus the high end performance improvement is scene in data loading processes.

A mandatory prerequisite before using the approach is that the internal tables are both sorted by the respective key fields.

### Sample Code & Demonstration

#### Nested Loop Using Parallel Cursor:

Following are the steps to implement parallel cursor

- (a) Sort the internal table
- (b) Read the internal it\_ekpo with key wa\_ekko-ebeln and use binary search. It will give the starting index of PO 123 in table it\_ekpo.
- (c) Loop on table it\_ekpo from index got from step2 and also put an exit condition of inner loop. Control will transfer to outer loop as exit condition passes.

```
CODE
DATA : it_ekko TYPE STANDARD TABLE OF ekko,
      wa_ekko TYPE ekko,
      it_ekpo TYPE STANDARD TABLE OF ekpo,
      wa_ekpo TYPE ekpo.
DATA : lv_tabix TYPE sy-tabix.
SELECT * FROM ekko INTO TABLE it_ekko.
```

```

SELECT * FROM ekpo INTO TABLE it_ekpo.
SORT it_ekko BY ebeln.
SORT it_ekpo BY ebeln.
LOOP AT it_ekko INTO wa_ekko.
  READ TABLE it_ekpo TRANSPORTING NO FIELDS WITH KEY ebeln = wa_ekko-ebeln binary
  search.
  IF sy-subrc EQ 0.
    lv_tabix = sy-tabix.
    LOOP AT it_ekpo INTO wa_ekpo FROM lv_tabix.
      IF wa_ekpo-ebeln NE wa_ekko-ebeln.
        EXIT.
      ENDIF.
      Logial ABAP statements .....
    ENDLOOP.
  ENDIF.
ENDLOOP.

```

Here when WA\_EKKO-EBELN =123 the inner loop will execute 4 times and statement inside the loop will execute 3 times. This will result in a major performance improvement over the nested loop code elaborated earlier in the document.

## Performance Data Comparison

Parallel cursor technique comes in very handy in SAP BI to optimize the data source extraction process and write high performance transformational routines. A comparative study in a real time project scenario with complex business logic defined to populate an enhanced & a newly added field in the following data sources showcased the high levels of performance improvement that can be gained using parallel cursor approach. For illustrative purposes, comparisons related to data load time of 3 data sources with enhanced fields and code defined in the corresponding exit is as below:

DS	Normal Nested Loop		Nested Loops with parallel cursor	
	No of records	Total Data load Time	No of records	Total Data load Time
0COORDER_ATTR	185748	45m 6s	185748	4m 6s
0CS_ORDER_ATTR	54522	24m 23s	54522	2m 53s
0PM_ORDER_ATTR	132227	22m 44s	132227	4m 22s

As the tabular detail above demonstrates, using parallel cursor technique we can achieve high level of performance optimization in SAP BI dataloading processes. The improvement in performance is far more pronounced when the volume of the data is increased.

## Related Content

[ABAP Code for Parallel Cursor - Loop Processing](#)

[Parallel Cursor Technique](#)

Book - > ABAP Development for SAP BW by Dirk Herzog

For more information, visit the [Business Intelligence](#) homepage.

## Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.