IBM SAP Technical Brief

Tuning SAP® with DB2® on AIX®

Mark Gordon
IBM Solutions Advanced Technical Skills

Version: 1.4
Date: August 31, 2012
1. ACKNOWLEDGEMENTS ................................................................................................................ 5

1.1. VERSION 1.0 ................................................................................................................................... 5

2. DISCLAIMERS ................................................................................................................................... 5

3. TRADEMARK ..................................................................................................................................... 5

4. FEEDBACK ......................................................................................................................................... 5

5. VERSION UPDATES .......................................................................................................................... 5

6. INTRODUCTION ................................................................................................................................6

7. ANALYZING A PROBLEM WITH A SPECIFIC PROGRAM OR TRANSACTION .............. 8

7.1. COMPONENTS OF SAP RESPONSE TIME .................................................................................... 8

7.2. DB2 CONCEPTS .............................................................................................................................. 9

7.2.1. Local and Join Predicates ..................................................................................................... 9

7.2.2. Explain ................................................................................................................................... 9

7.2.3. Estimates of Rows and Cost ................................................................................................. 12

7.2.4. Filtering Estimates ............................................................................................................... 13

7.2.5. Prefetch ................................................................................................................................1 3

7.3. MAJORITY OF ELAPSED TIME IS CPU ON APPLICATION SERVER ..................................................... 16

7.3.1. Performance problem is high CPU time .............................................................................. 16

7.3.1.1. Summary of Performance problem is high CPU time ................................................. 21

7.3.2. High CPU time caused by LOOP AT statement .................................................................. 21

7.4. MAJORITY OF TIME IS DATABASE REQUEST TIME ......................................................................... 24

7.4.1. Types of problems causing high DB request time ................................................................ 24

7.4.1.1. Indexes do not support predicates................................................................................... 24

7.4.1.2. Misuse of SAP Data Model ........................................................................................... 24

7.4.1.3. SELECT in LOOP instead of FOR ALL ENTRIES .................................................... 24

7.4.1.4. Data skew causes wrong access path to be chosen...................................................... 25

7.4.1.5. Invalid or missing DB2 statistics ..................................................................................... 25

7.4.1.6. Unnecessary SQL............................................................................................................... 25

7.4.1.6.1. Identical DB calls......................................................................................................... 25

7.4.1.6.2. Table could be buffered on application server but is not buffered .............................. 26

7.4.1.6.3. SAP instance buffers are too small .............................................................................. 26

7.4.1.6.4. FOR ALL ENTRIES with empty internal table .......................................................... 26

7.4.1.6.5. Invalid parameters in ABAP SQL call......................................................................... 26

7.4.2. Performance problem caused by identical DB calls............................................................ 26

7.4.2.1. Summary of performance problem caused by identical DB calls ................................ 30

7.4.3. Performance problem – empty itab with For All Entries .................................................... 30

7.4.3.1. Summary of Performance problem – empty itab with For All Entries .......................... 32

7.4.4. Performance problem caused by data skew ....................................................................... 33

7.4.4.1. Summary of Performance problem caused by data skew ............................................ 39

7.4.5. Performance problem - table buffering candidate............................................................... 40

7.4.5.1. Summary of Performance problem - table buffering candidate .................................. 43

7.4.6. Performance problem where predicates do not match indexes ........................................ 43
7.4.6.1. Summary of performance problem where predicates do not match index ............ 45
7.4.7. Runtime analysis of SQL when you cannot use ST05 .............................................. 45
7.4.8. Invalid parameters cause unnecessary SQL .............................................................. 50
7.5. MAJORITY OF TIME IS NOT CPU OR DB REQUEST TIME ......................................... 51

8. SYSTEM PERFORMANCE PROBLEMS ............................................................................ 51

8.1. PERFORM SQL CACHE ANALYSIS .............................................................................. 51
8.1.1. Indicators of inefficient SQL access – Netweaver 7.10 .............................................. 51
8.1.2. Indicators of inefficient SQL access – Netweaver 7.0 and earlier .............................. 54
8.1.3. Index-Only Access ................................................................................................. 56
8.1.4. Performance problem - local predicates do not match indexes ............................... 59
8.1.4.1. Summary of Performance problem - local predicates do not match indexes ......... 65
8.1.5. Reasons that the predicates do not match indexes ................................................... 65
8.1.6. Actions when local predicates do not match indexes ............................................ 65
8.1.7. DB2 Index Advisor in ST04 ..................................................................................... 66
8.1.7.1. Why one should be cautious with index advisor .................................................. 71
8.1.8. Performance problem - incorrect use of SAP data model ....................................... 71
8.1.8.1. Summary of Performance problem - incorrect use of SAP data model ............... 74
8.1.9. Performance problem - Indexes do not support join predicates ............................... 74
8.1.10. Performance problem SAP buffer space ............................................................... 78
8.1.10.1. Summary of symptom of SAP buffer problem .................................................. 82
8.1.11. Performance problem – missing index on custom table ........................................ 83
8.1.12. Performance problem – Slow SQL due to LOB columns ....................................... 84

8.2. CREATE CANDIDATE LIST WITH ST04 TABLES ..................................................... 87
8.2.1. ST04 Tables and TEMP table activity ..................................................................... 88

8.3. CREATE CANDIDATE LIST FROM ST03N ................................................................. 93
8.4. ST03N WITH TABLE STATISTICS ............................................................................. 95

9. SYSTEM HEALTH CHECK ............................................................................................. 95

9.1. CPU ACTIVITY ............................................................................................................. 95
9.2. I/O ACTIVITY .............................................................................................................. 99
9.2.1. Good I/O performance in DB2 .............................................................................. 99
9.2.2. Symptom of I/O hotspot on disk ......................................................................... 99
9.2.3. Symptom of slow I/O ......................................................................................... 99
9.3. DB2 REVIEW .............................................................................................................. 100
9.3.1. Database Hit Rate ................................................................................................. 100
9.4. SAP BUFFER SETTINGS ......................................................................................... 102
9.5. SAP BUFFERED TABLE STATISTICS ..................................................................... 103
9.5.1. Not buffered but could be buffered ...................................................................... 103
9.6. EVALUATE MEMORY IN AIX ................................................................................... 105
9.6.1. AIX 6.1 Restricted Tunables concept ................................................................. 105
9.6.2. AIX Alternative Memory Management .............................................................. 105
9.6.3. AIX paging ......................................................................................................... 106
9.6.4. Evaluating increasing memory for DB2 or SAP .................................................. 109

© Copyright IBM Corporation, 2012
10. FOUR GUIDELINES FOR AVOIDING PERFORMANCE PROBLEMS............................... 111

10.1. USE THE SAP DATA MODEL .................................................................................................... 111
10.2. USE ARRAY OPERATIONS ON THE DATABASE........................................................................... 112
10.3. CHECK WHETHER THE DATABASE CALL CAN BE AVOIDED ....................................................... 112
10.4. WRITE ABAP PROGRAMS THAT ARE LINE-ITEM SCALABLE .................................................... 112

11. APPENDIX 1: SUMMARY OF PERFORMANCE MONITORING TOOLS.......................... 113

11.1. SAP ........................................................................................................................................ 113
11.1.1. DBACOCKPIT................................................................................................................... 113
11.1.2. DB02 Transaction.............................................................................................................. 113
11.1.3. SE11 Transaction............................................................................................................... 113
11.1.4. SE30 Transaction............................................................................................................... 113
11.1.5. SM12 Transaction............................................................................................................... 113
11.1.6. SM50 Transaction.............................................................................................................. 114
11.1.7. SM51 Transaction.............................................................................................................. 114
11.1.8. SM66 Transaction.............................................................................................................. 114
11.1.9. STAD Transaction.............................................................................................................. 114
11.1.10. ST02 Transaction............................................................................................................... 114
11.1.11. ST03N Transaction ........................................................................................................ 114
11.1.12. ST04 Transaction............................................................................................................... 115
11.1.13. ST05 Transaction............................................................................................................... 115
11.1.14. ST06N Transaction ........................................................................................................ 115
11.1.15. ST10 Transaction............................................................................................................... 115
11.1.16. RSINCL00 Program....................................................................................................... 115
11.1.17. SQLR Transaction.......................................................................................................... 115
11.1.18. RSTRC000 Program ...................................................................................................... 115
1. Acknowledgements

1.1. Version 1.0

Thank you to Damir Rubic and Steven Poon, who contributed material on memory management, and also reviewed the paper and offered suggestions for changes and improvements.

2. Disclaimers

IBM has not formally reviewed this paper. While effort has been made to verify the information, this paper may contain errors. IBM makes no warranties or representations with respect to the content thereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. IBM assumes no responsibility for any errors that may appear in this document. The information contained in this document is subject to change without any notice. IBM reserves the right to make any such changes without obligation to notify any person of such revision or changes. IBM makes no commitment to keep the information contained herein up to date.

Most of the examples in this paper are from ECC 6.0 and DB2 9.7 or 9.5. A few examples are from earlier versions of SAP or DB2.

The processes and guidelines in this paper are the compilation of experiences analyzing performance on a variety of SAP systems. Your results may vary in applying them to your system.

Examples have been edited and graphics have been modified to create or clarify examples for the paper.

3. Trademark

- SAP®, R/3®, ABAP™ and SAP NetWeaver® are the trademark(s) or registered trademark(s) of SAP AG in Germany and in several other countries.
- IBM®, RS/6000®, System p®, pSeries®, DB2® and AIX® are registered trademarks of IBM Corporation.

4. Feedback

Please send comments or suggestions for changes to gordonmr@us.ibm.com.

5. Version Updates

- Version 1.0 – initial version
- Version 1.1 – add Section 8.1.1.
- Version 1.2 – Add Section 7.2, Section 8.1.7, and Section 8.1.9, and various corrections.
- Version 1.3 – Add Sections 7.2.5, 7.4.6, 8.2.1 and 8.1.12 and various corrections.
- Version 1.4 – Add Section 7.4.8 and update Section 7.3.1 to use SAT.
6. Introduction

There are two intended audiences for this paper – DB2® DBAs and SAP BASIS administrators. Either may be doing performance analysis on an SAP system with DB2 on AIX. The goal of this paper is to provide each audience with material that is useful and new: An SAP Basis administrator experienced with other databases should find the DB2 specific tuning tools and techniques helpful, while the experienced DB2 administrator is presented with SAP specific tuning tools and techniques.

This paper covers the two most common types of performance problems – database performance and inefficient ABAP coding. While there are other causes of problems in SAP (e.g. network performance, external RFC interfaces, SAP instance configuration, SAP sort, etc), database and ABAP performance are the most common and generally have the biggest impact. In order to provide the most benefit in the smallest paper, these other issues are not included in this paper.

This paper has a process-based approach, where different goals are pursued via different processes and tools.

- **To fix a problem reported for a specific program**, we will perform elapsed time analyses of programs, determine where time is spent, and optimize these long running parts. This includes interpretation of ST03N and STAT/STAD records, and using ST05 and SE30. The paper will demonstrate how to use the SAP stats to obtain database performance statistics, identify I/O bottlenecks and SAP problems, etc. The benefit of this approach is that it is focused on an area that has been identified as a business problem.

- **To check for inefficient use of DB resources and improve overall database server performance**, we will use ST04 statement cache analysis. The value of this approach is that it offers a very big potential payoff in reducing resource usage and increasing system efficiency. The disadvantage is that one may be finding and solving problems that no end-user cares about. For example, if we can improve the elapsed time of a batch job from 2 hours to 10 minutes, but the job runs at 2:00 AM, and nobody needs the output until 8:00 AM, it may not really be a problem. Even if it is not a business problem, it may still be beneficial to address a problem of this type as part of optimizing resource consumption, in order to reduce the computing resources required to support business requirements.

- **To do a system health check**, review AIX paging and CPU usage, DB2 I/O statistics, ST10 and ST02 buffering. The operating environment needs to be running well for good performance, but problems in these areas can be symptoms of other problems. For example, inefficient SQL can cause high CPU usage or high I/O activity. Therefore, a health check should be done together with analysis of SQL and ABAP problems.

This paper has many examples, and it describes what is good or bad in each example. There are not always specific rules given on what is good or bad, such as “Database request time” over 40% of “elapsed time” is bad and less than 40% is good. Rather, this paper tries to focus on an opportunity-based approach, such as:

- Look for where a program (or the SAP and database system) spends time.
- Ask “If I fix a problem in this area, will people notice and care that it has been fixed?”

© Copyright IBM Corporation, 2012
It will discuss how to estimate the impact of solving a problem. System wide performance analysis (such as a statement of cache analysis, or ST03 analysis) will generally turn up several candidates. By estimating the impact of fixing these problems, one can decide which to address first.

When doing this analysis, it is important to identify and track specific issues. Often, a performance issue may not have enough impact to merit a new index, or an ABAP change. In this case, we want to track that we have analyzed it, and chosen not to do anything, so that we don’t waste time discovering it again next year.

This paper refers to a number of SAP Notes. An OSS userid, or userid that allows access to service.sap.com, is a prerequisite for anyone doing performance analysis on an SAP system, whether the person is a DB2 DBA, AIX administrator, or SAP BASIS Administrator.

This paper breaks performance management into three parts, which are discussed in Sections 7, 8, and 8.4:

- Analyzing a problem with a specific program or transaction (Section 7)
- System performance problems (Section 8)
- System Health Check (Section 8.4)

Since AIX-level symptoms such as paging, excessive CPU use, and high I/O rates can be symptoms of application and SQL problems, one needs to start with reviewing the SAP and DB2 indicators, before taking action (such as adding memory or CPU) based on the AIX level indicators.
7. Analyzing a problem with a specific program or transaction

If a problem has been identified, then the first step is to determine where most of the response time is spent. Based on this first step, there are different tools that are used to monitor the ABAP, database, RFCs, etc.

7.1. Components of SAP response time

SAP note 364625 describes the different components of SAP dialog step response time. This paper is focused on programs that have high database request time, or high CPU time. These are the two most frequent causes of performance problems. For information on more detailed analysis of other components of response time, see document WP100287 at www.ibm.com/support/techdocs.

Figure 1 is the output of the STAD transaction, which is used to display total response time and components of response time (DB, CPU, etc) for an individual dialog step.

![Figure 1: STAD transaction display of time components](image)
7.2. **DB2 Concepts**

Here, we will have a brief review of some of the concepts and terms that will be used for DB2.

### 7.2.1. Local and Join Predicates

Local predicates on columns qualify the rows to be returned to the application. Join predicates specify the relationship between tables used to join rows in the tables.

![Sample SQL with Join and Local Predicates](image)

In Figure 2, \( T_{00}."ATINN" = ? \) is a local predicate, and \( T_{00}."ATINN" = T_{01}."ATINN" \) is an example of a join predicate.

### 7.2.2. Explain

We can explain SQL to evaluate whether it is executed efficiently. The explain output will provide some clues that we can use to determine whether the SQL can be improved.

![Display Execution Plan for SQL Statement](image)

Figure 3: BKPF Explain Sample

© Copyright IBM Corporation, 2012
In Figure 3, there are two local predicates (MANDT = ? and BELNR = ?) but “key columns” is 1. This means that only the local predicate on the first column is index matching (processed in the index b-tree). If we select the “BKPF~0” field in the explain, and press the “Details” button, we will see how DB2 processes the local predicates.

<table>
<thead>
<tr>
<th>Predicates:</th>
</tr>
</thead>
<tbody>
<tr>
<td>relational operator:</td>
</tr>
<tr>
<td>subquery input required:</td>
</tr>
<tr>
<td>filter factor:</td>
</tr>
</tbody>
</table>

**Predicate text:**

(M1.MANDT = ?)

**Predicate text:**

(M1.BELNR = ?)

Figure 4: Start/Stop and Sargable local predicates

Details shows that MANDT is used as a start and stop condition. This means that it is processed in the index b-tree, which is the most efficient access. BELNR is Sargable in the index, which means that DB2 has to check each index entry and compare its value to the value passed from the program. This is less efficient than index matching (start/stop) access.

If we drill into the BKPF table name in Figure 3, we will see the indexes defined on the table. Here we can see why BELNR is SARGable – there is a column in the index before BELNR (BUKRS) that has no local predicate. DB2 can use index matching access only on concatenated columns starting with the first columns. If there are columns in an index that have no local or join predicate, then local predicates on subsequent index columns will be SARGable.

| PRIMARY KEY | BKPF-0 |
|-------------|
| Column Name | # Distinct |
| MANDT       | 1         |
| BUKRS       | 24        |
| BELNR       | 1,851,606 |
| GJAHN       | 1         |

Figure 5: BKPF~0 index

© Copyright IBM Corporation, 2012
This SQL can be improved if the ABAP developers can modify the program to add a selection on BUKRS. If this can be done, the SQL will run much more quickly, since DB2 can then process the local predicates on MANDT, BUKRS, and BELNR in the index b-tree.

**Figure 6: BKPF Explain after improvement**

In Figure 6, note that “#key columns” is now 3 after adding “BUKRS = ?” to the SQL. This SQL will run much faster than the SQL in Figure 3. If we select “BKPF~0” and press “Details”, we see that now all three local predicates are index matching (Start/Stop).

### Predicates:

<table>
<thead>
<tr>
<th>4 ) Used as Start Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>relational operator:</td>
</tr>
<tr>
<td>subquery input required:</td>
</tr>
<tr>
<td>filter factor:</td>
</tr>
</tbody>
</table>

**Predicate text:**

(QL.MANDT = ?)

<table>
<thead>
<tr>
<th>3 ) Used as Start Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>relational operator:</td>
</tr>
<tr>
<td>subquery input required:</td>
</tr>
<tr>
<td>filter factor:</td>
</tr>
</tbody>
</table>

**Predicate text:**

(QL.BUKRS = ?)

<table>
<thead>
<tr>
<th>2 ) Used as Start Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>relational operator:</td>
</tr>
<tr>
<td>subquery input required:</td>
</tr>
<tr>
<td>filter factor:</td>
</tr>
</tbody>
</table>

**Predicate text:**

(QL.BELNR = ?)

**Figure 7: BKPF Details after improvement**
7.2.3. Estimates of Rows and Cost

In the explain, DB2 gives estimates of the cost of performing an operation, as well as the number of rows processed at each step. Note how in Figure 8, the estimate of the number of rows retrieved is low (2.01, that is $2.01 \times 10^0$) and the cost is several orders of magnitude greater ($5.36 \times 10^4$). When $\text{tot\_cost}$ is much higher than $\text{num\_rows}$, it is also a clue that we need to investigate the SQL to see if it is being executed efficiently.

<table>
<thead>
<tr>
<th>0</th>
<th>SELECT STATEMENT</th>
<th>(Estimated Costs = 5.366E+04 [timerows])</th>
<th></th>
<th>num_rows</th>
<th>tot_cost</th>
<th>1/o_cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RETURN</td>
<td>2.010E+00</td>
<td>5.366E+04</td>
<td>1.171E+04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>FETCH</td>
<td>2.010E+00</td>
<td>5.366E+04</td>
<td>1.171E+04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>IXSCAN</td>
<td>#key columns: 1</td>
<td></td>
<td>2.010E+00</td>
<td>5.366E+04</td>
<td>1.171E+04</td>
</tr>
</tbody>
</table>

Figure 8: High $\text{tot\_cost}$ and low $\text{num\_rows}$

<table>
<thead>
<tr>
<th>0</th>
<th>SELECT STATEMENT</th>
<th>(Estimated Costs = 2.273E+01 [timerows])</th>
<th></th>
<th>num_rows</th>
<th>tot_cost</th>
<th>1/o_cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RETURN</td>
<td>1.000E+00</td>
<td>2.273E+01</td>
<td>3.002E+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>FETCH</td>
<td>1.000E+00</td>
<td>2.273E+01</td>
<td>3.002E+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>IXSCAN</td>
<td>#key columns: 3</td>
<td></td>
<td>1.000E+00</td>
<td>1.514E+01</td>
<td>3.000E+00</td>
</tr>
</tbody>
</table>

Figure 9: Low $\text{tot\_cost}$ and low $\text{num\_rows}$

In Figure 9, $\text{tot\_cost}$ and $\text{num\_rows}$ are closer together, so DB2 is estimating that the statement is efficient in retrieving rows.

The DB2 estimates are just that – estimates based on the information available to DB2. The estimates can be very different from actual row counts in situations where RUNSTATS are missing or out of date, or there is skew in the data and SAP is preparing the statements with parameter markers. See the example with the impact of prepare with parameters and skew in Section 7.4.4.
7.2.4. Filtering Estimates

When possible, we want DB2 to be able to process filtering local predicates in an index. DB2 shows filtering estimates in the Explain Details.

<table>
<thead>
<tr>
<th>Predicates:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 ) Used as Start Condition</td>
</tr>
<tr>
<td>relational operator:</td>
</tr>
<tr>
<td>subquery input required:</td>
</tr>
<tr>
<td>filter factor:</td>
</tr>
<tr>
<td>predicate text:</td>
</tr>
<tr>
<td>9 ) Used as Stop Condition</td>
</tr>
<tr>
<td>relational operator:</td>
</tr>
<tr>
<td>subquery input required:</td>
</tr>
<tr>
<td>filter factor:</td>
</tr>
<tr>
<td>predicate text:</td>
</tr>
<tr>
<td>2 ) Sargable Predicate</td>
</tr>
<tr>
<td>relational operator:</td>
</tr>
<tr>
<td>subquery input required:</td>
</tr>
<tr>
<td>filter factor:</td>
</tr>
<tr>
<td>predicate text:</td>
</tr>
</tbody>
</table>

Figure 10: Details filter factors

A filter factor of 1 means that DB2 estimates that the local predicate will qualify all rows, that is that all rows would be selected. A small filter factor means that DB2 estimates that a small number of rows will qualify, for example 5.4 * 10^-7 means that DB2 estimates that one 1 row in 1.8M will be selected (1/1800000 = 5.4 * 10^-7)

7.2.5. Prefetch

When DB2 estimates that many rows will be retrieved, it may use prefetch. This will read many pages of data with one operation, and can give a significant performance improvement compared to not using prefetch.
Here is an example using SQL from the SAP FAGLL03 transaction where DB2 does not use prefetch.

```
0 SELECT STATEMENT
  1 RETURN 5.1358E-04 5.4728E+04 7.2324E+03
  2 NLJOIN 6.1380E-04 5.4728E+04 7.2324E+03
  3 [0] TBSSCAN GENROV 1.0300E+00 1.3943E-04 0.0000E+00
  4 [1] HSJOIN 5.1380E-04 5.4728E+04 7.2324E+03
  5 [0] FETCH BSIS 1.4997E+04 3.8325E+04 5.1457E+03
    6 ISCAN BSIS=0 #key columns: 3 1 4997E+04 1 8843E+03 1 4257E+02
  7 [1] FETCH FAGLFLX=2 2.2228E+03 1.5783E+04 2.0857E+03
    8 ISCAN FAGLFLX=0 #key columns: 4 2.2228E+03 2.1622E+02 2.5833E+01
```

**Figure 11: FAGLL03 explain**

In Figure 11, select a FETCH line and press “Details”

**Figure 12: FETCH details without prefetch**

This access path is not using prefetch.

© Copyright IBM Corporation, 2012
Figure 13: FAGLL03 with prefetch

In Figure 13, DB2 estimates that many more rows will be retrieved than in the example in Figure 11 – compare num_rows for the IXSCAN operations. Here, the SORT and RIDSCN are part of DB2 access path that prepares for prefetch on the FETCH operator. If you place the cursor on FETCH and press “Details”, you will see LIST prefetch being used.

Figure 14: LIST prefetch

It is important to note that one cannot directly control whether prefetch is used. DB2 will choose it if the DB2 optimizer considers it the best way to execute the SQL. As shown later in this paper in the example in Section 7.4.4, the better the cost estimate, the better the access path that DB2 chooses. In this example, as in Section 7.4.4, preparing the statement with values enabled DB2 to choose a better access path than the default preparing with markers.
7.3. **Majority of elapsed time is CPU on application server**
When the majority of time is CPU on the application server, we will use SAT (formerly SE30) to analyze the ABAP program.

7.3.1. **Performance problem is high CPU time**
As an example, assume that we have been asked to investigate the performance of ZSUMMARY. We check ST03N as shown in Figure 15.

![Figure 15: ST03N ZSUMMARY](image)

Average times are somewhat slow ("0 Time" is 174 seconds per dialog step), with CPU on the application server over ten times length of database request time.

Since CPU time is the majority of elapsed time in the example in Figure 15, use SAT to trace the ABAP runtime. We arrange for a user to run ZSUMMARY so we can trace it.
Run SAT and click the “Own Standard Variant” button and “Change”.

**Figure 16: SAT own standard variant selected**

Then press “change” (the pencil) in Figure 16 to change the “own standard variant” options to set default settings.

**Figure 17: SAT gather stats on internal tables**

© Copyright IBM Corporation, 2012
Enable statistics for internal tables as shown in Figure 17. By default these are off, but they are the most common source of high CPU use in ABAP programs.

In the “Duration/Type” tab, select “Per Call Position”.

Figure 18: SAT Aggregation level

In Figure 18, select Aggregation level by call, so that different calls on the same table are reported separately. Aggregation level “None” will cause the trace to grow very big very quickly, but is used if you need to determine the hierarchy relationships between calling and called programs.

Press “Save” on the screen in Figure 18 and green arrow to go back to the SAT main screen shown in Figure 16. Then press “Switch On/Off” on the SAT main screen to show the list of work processes here in Figure 19.

Figure 19: SAT start/end measurement
Select the process to be traced, and press “Start measurement”, trace a while, then press “End measurement”.

Return back to the main SAT screen shown in Figure 16. Press “Format performance data”, and select trace to be formatted.

![ABAP Runtime Analysis: Initial Screen](image)

**Figure 20: SAT analysis**

After the trace is formatted, evaluate it.

![ZSUMMARY Desktop](image)

**Figure 21: ZSUMMARY Desktop**

As shown in Figure 21, the program spent over 98% of elapsed time processing Internal Tables on the application server.

Press the “Times” tab and then sort the list by “Net” to see where the time was spent.

![SAT sorted by Net time](image)

**Figure 22: SAT sorted by Net time**

© Copyright IBM Corporation, 2012
Select the slow “Read Table” statement in Figure 22 and press ‘ABAP’ to see the ABAP.

```
0077, 0868, 0869:
READ TABLE gt_wflog INTO go_wflog WITH KEY
  objkey = lv_objkey.
```

**Figure 23: SAT source code**

This is custom code – it starts with Z. We will send it to the developers to fix it.

You can calculate the time per READ TABLE using the “Hit Lists” button in Figure 20.

**Hit List for Single Trace**

<table>
<thead>
<tr>
<th>Event Class</th>
<th>Event Type</th>
<th>Event Name</th>
<th>Code</th>
<th>Net 2</th>
<th>No. 2</th>
<th>Gross 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>C Itab</td>
<td>Read Table</td>
<td>IT2(000001)</td>
<td>T2</td>
<td>308,657,854</td>
<td>6,645</td>
<td>308,657,854</td>
</tr>
<tr>
<td>C Itab</td>
<td>Read Table</td>
<td>IT2(000001)</td>
<td>T2</td>
<td>8,181,504</td>
<td>1,929</td>
<td>8,181,504</td>
</tr>
<tr>
<td>C Itab</td>
<td>Read Table</td>
<td>IT2(000001)</td>
<td>T2</td>
<td>5,290,092</td>
<td>1,911</td>
<td>5,290,092</td>
</tr>
<tr>
<td>D Store Int</td>
<td>Select Single</td>
<td>STXH</td>
<td>T2</td>
<td>3,994,425</td>
<td>1,929</td>
<td>3,994,425</td>
</tr>
</tbody>
</table>

**Figure 24: Hit Lists**

In Figure 24, 308,697,854 uSec / 9645 calls = 32,005 uSec (32 ms) per call, which is very slow.

Performance problems reading internal tables are very common performance and scalability issues. In the SAT transaction, press the “Tips and tricks” button to see various suggestions on ways to improve ABAP programming. “Tips and tricks” contains several examples of common problems with internal tables.

**Figure 25: SAT tips and tricks**

© Copyright IBM Corporation, 2012
One can review “Tips and tricks” for suggestions on ways to improve ABAP programs. In this case, the problem appears to be that the ABAP is doing a linear search of the internal table.

<table>
<thead>
<tr>
<th>Linear search in an internal table</th>
<th>Binary search in an internal table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entries: 1000, Line width: 100</td>
<td>Entries: 1000, Line width: 100</td>
</tr>
<tr>
<td>* Key width: 20</td>
<td>* Key width: 20</td>
</tr>
<tr>
<td>The READ ends with ST-SUERC=4</td>
<td>The READ ends with ST-SUERC=4</td>
</tr>
<tr>
<td>READ TABLE ITAB INTO UA</td>
<td>READ TABLE ITAB INTO UA</td>
</tr>
<tr>
<td>WITH KEY K = 'X'</td>
<td>WITH KEY K = 'X'</td>
</tr>
<tr>
<td></td>
<td>BINARY SEARCH.</td>
</tr>
</tbody>
</table>

Figure 26: SAT binary search test

When a large internal table is read without the ‘BINARY SEARCH’ option, the program becomes much slower as the number of items processed grows and the table grows.

7.3.1.1. Summary of Performance problem is high CPU time

Check the response time components of the program. If a majority of time is spent in CPU time on the application server, use SAT to trace the program. We find that the majority of ABAP time is on a READ TABLE statement.

This is a custom program, so we would send this to the development team to investigate ways to speed up processing the internal tables, such as using a sorted itab with “BINARY SEARCH” option on READ TABLE.

7.3.2. High CPU time caused by LOOP AT statement

Another common cause of high CPU time for SAP transactions is in processing LOOP AT statements on internal tables. Similar to the READ TABLE example in Section 7.2, LOOP AT performance can be improved by using SORTED internal tables.

© Copyright IBM Corporation, 2012
In Figure 27, we have traced execution of RPCALCU0, and find the vast majority of elapsed time is ABAP. The average call times for the LOOP AT statements are long, for example 18,386,505 microseconds / 26 calls = 707 ms per call, which is very long. When we display the ‘hit list” for the trace in Figure 28, we see that most of the time is in LOOP AT statements on internal tables. LOOP AT time is CPU on the application server.

When we display the ABAP source for a LOOP AT IT_719 statement in Figure 28, we find that the problem is in SAP code in Figure 29. We would use “Goto > Attributes” in Figure 29 to see who created the ABAP.
In this case, we verified that the program was created and last changed by SAP, so we search SAP Service Marketplace and find that there is a SAP note that fixed the problem by using a sorted internal table.

Figure 29: RPCACLU0 ABAP

Figure 30: RPCACLU0 sorted itab note 583807

© Copyright IBM Corporation, 2012
7.4. **Majority of time is Database Request time**

If the majority of program or transaction elapsed time is DB time, the first question we need to ask is whether the DB time is high because calls to the DB are slow, or for some other reason. There are a number of different problems that manifest themselves as high DB request time.

7.4.1. **Types of problems causing high DB request time**

7.4.1.1. **Indexes do not support predicates**

Local predicates are the “column operator value” clauses in SQL. In order to execute these efficiently, the columns referenced in the local predicates must be in an index. If the column is not not in an index, then DB2 must read the table to check whether the row contains the value which is sought.

Join predicates are the `table1.columnA = table2.columnB` clauses in SQL. Nested Loop Join (NLJ) is the most common join method used with SAP, and in order to process an efficient NLJ, the inner table must have an index containing the columns referenced in the join (or local) predicates of the SQL.

There are examples of this in Section 8.1.4 and Section 7.4.6.

7.4.1.2. **Misuse of SAP Data Model**

SAP often stores data redundantly in more than one table. For example, a delivery document might contain the order number. And an order document might contain the associated delivery number. But, the delivery tables are indexed by delivery number, and order tables are indexed by order number. If a program has the value of an order number and wants to retrieve the associated delivery, the program could use either table, since both tables contain both columns. But, if the program selects from the delivery table “where order_number=”, there is no index for order number and the select is slow. If the program selects from the order table “where order_number=”, there is an index and the select is fast.

There are three SAP notes (185530, 191492, and 187906) that describe common errors in use of the data model and how to fix them.

There is an example of this problem in Section 8.1.8 and Section 8.1.9.

7.4.1.3. **SELECT in LOOP instead of FOR ALL ENTRIES**

If an ABAP program has a list of keys in an internal table, there are two ways to use the table to select rows from the database:

```
LOOP AT internal_table
SELECT … from DB_TABLE where COLUMN = internal_table-column
ENDLOOP
```
Each SELECT call in the LOOP will make a call between the application server, and the database server. Instead, an array operation should be used:

```
SELECT ... from DB_TABLE for all entries in internal_table
WHERE COLUMN=internal_table-column
```

The LOOP AT will make one call to the DB for each value, the FOR ALL ENTRIES will group the values together and make fewer DB calls.

### 7.4.1.4. Data skew causes wrong access path to be chosen

With the default RUNSTATS gathered by SAP, when DB2 estimates the number of rows that will be returned on a select, it uses the column cardinality and table cardinality to estimate the number of rows with each column value. For example if a column AUFNR has 1,000 distinct values, and its table has 10,000 rows, then DB2 calculates that each value specified for AUFNR will return 10 rows on the average.

However, if many rows contain the same value and few rows contain other values, then the data is skewed -- that is the distribution of values in rows is not uniform. If data is skewed, either DB2 requires additional histogram statistics combined with ABAP HINT, in order to use values at execution time to determine the skew, or DB2 needs an “optimization guideline” in the ABAP to specify the access path.

There is an example of this in Section 7.4.4.

### 7.4.1.5. Invalid or missing DB2 statistics

Cost based optimizers collect statistics about tables and indexes (number of rows, distinct values of columns, etc) that are used to optimize SQL statements. If the statistics are missing, or not collected in the correct way, the optimizer may choose the wrong access path.

### 7.4.1.6. Unnecessary SQL

#### 7.4.1.6.1. Identical DB calls

Due to program structure errors, the ABAP may repeatedly retrieve the same rows over and over, or update the same rows over and over. ST05 can be used to identify identical selects.

There is an example of this issue in Section 7.4.2.
7.4.1.6.2. Table could be buffered on application server but is not buffered
Tables that are seldom changed, and where the application can tolerate a small interval between when rows are changed and the changes are available on all application servers can be buffered on the SAP application server to offload the database.

There is an example of this issue in Section 7.4.5.

7.4.1.6.3. SAP instance buffers are too small
If the SAP generic or single record buffers are too small, then tables that could be buffered on the application cannot fit into buffer and must be read from the DB server.

There is an example of this problem in Section 8.1.10.

7.4.1.6.4. FOR ALL ENTRIES with empty internal table
The ABAP FOR ALL ENTRIES statement uses an internal table with a list of keys to be retrieved for a column or columns. If the internal table is empty, the local predicates on the columns referencing the internal table are not generated in the SQL, which generally causes the program to retrieve rows that it does not need.

There is an example of this problem in Section 7.4.3.

7.4.1.6.5. Invalid parameters in ABAP SQL call
If the ABAP does not correctly validate the inputs and SQL selections at runtime, it is possible to have invalid parameters in the DB calls - selection conditions that would never retrieve a row.

There is an example of this problem in Section 7.4.8.

7.4.2. Performance problem caused by identical DB calls
We have been asked to review the performance of ZMM02. Since the transaction starts with a Z, we know it is custom code – the customer namespace starts with Z or Y. However, it may be calling SAP standard ABAP routines, so we will not know if the issue is in SAP or custom ABAP until we trace it.

Figure 32 is an excerpt from ST03N, and shows the main components of response time for this transaction. Average response time is 18.5 seconds (“0 Response Time” is in ms), and Average DB call time (0 DB Time is 15,741 ms) is about 10 times as large as average CPU on the application server (1,449 ms). So, we will use ST05 trace to review the reason for the long average DB times.
Figure 32: ZMM02 in ST03N

Figure 33: ZMM02 STAT record

Figure 33 is a STAT record for a single execution of ZMM02. We can retrieve the individual STAT records using the STAD transaction. Note that while sequential read is the majority of...
DB call time, the average time per row (0.8 ms) is fast, so the long DB time is not caused by slow calls to the DB, but by the number of calls to the DB.

We trace with ST05 and list the trace, as shown in Figure 34. Reviewing the SQL calls in this trace, it appears that the same information is being repeatedly retrieved – note the values specified in the local predicates.

Figure 34: ZMM05 ST05 trace

We can check for duplicated SQL calls in the trace. Use “Trace List > Display Identical Selects” to see the report in Figure 35. Here, we see that the same data is being repeatedly retrieved from the MVER table, which confirms what we saw in the formatted trace in Figure 34.

Figure 35: ZMM02 Identical Selects

We can format the trace in Figure 34 in a different way, to evaluate how much time the MVER calls take. Starting from the trace in Figure 34, use “Trace List > Summarize by SQL statement” to see the aggregated time per statement shown in Figure 36.
The summarized trace shows that the MVER calls are about 90% of total DB time in the trace. 89% of MVER calls are identical (using same parameters as another call). The last step is to determine which program is making the calls. From the trace in Figure 34, click on one of the MVER OPEN lines, and press the “Display call position in ABAP program” button (looks like white sheet of paper), to see which program made the calls to DB2. The program is custom code (it starts with Z).

And we report this to the ABAP team, to review the cause for the repeated identical calls. Since the FORM does not have a loop making the SELECT SINGLE, the logic problem is in the program calling the FORM, which could be determined using SE30 ABAP trace of ZMM02.
Similar to the problem described in Section 7.2, the root cause of this performance is in the ABAP code. However, this ABAP problem manifests itself as high DB time, while the ABAP problem in Section 7.2 caused high CPU time on the application server.

### 7.4.2.1. Summary of performance problem caused by identical DB calls

ZMM02 DB call time is about 10x CPU on the application server. STAT record showed that average call times are good (less than one ms), so the cause of the long DB time is the number of calls, not slow DB calls. We trace the program with ST05, and process the trace to see that the same data is repeatedly retrieved from DB2. The table with the identical calls (MVER) makes up about 90% of the DB call time. Since the program is custom ABAP program (starts with Z), the development team will do further analysis to determine the cause of the identical DB calls from FORM f_monthly_consumption.

A problem that looked at first like a DB performance issue (DB time much higher than CPU time) is actually an ABAP structure issue.

### 7.4.3. Performance problem – empty itab with For All Entries

We’ve been asked to review the performance of ZPUR11. The average response time is 26 seconds (0 Time), and about 25 seconds of that time is DB call time (0 DB Time).

---

Figure 38 : ZPUR11 ST03N

So, we use ST05 to trace the execution.

---

Figure 39: ZPUR11 - many rows on EINE

© Copyright IBM Corporation, 2012
And find relatively long selects (over a second) on EINE that select more rows than can be displayed in ST05. Looking at the SQL, we notice that there is only one local predicate – MANDT = ‘200’. Since there is generally on only one productive MANDT in a SAP system (there is only one MANDT value here), this means that the entire EINE table is being retrieved. Retrieving all the rows of a table is unusual. Press the “Explain” button and then drill into the blue EINE table name in Figure 40 to display the explain, along with table and index data.

Figure 40: EINE explain with table data

We can format the trace in Figure 39 using “Trace List > Summarize trace by SQL statement”, and we see that the EINE selects are roughly half of DB time. And each select retrieves ~166K rows. Why could the application repeatedly read the entire EINE table?

Figure 41: ZPUR11 summarized SQL trace
© Copyright IBM Corporation, 2012
We can display the ABAP source, by selecting the EINE line in Figure 41, and pressing “Display call positions in ABAP programs” (the button that looks like sheet of paper).

Figure 42: ZPUR11 ABAP with For All Entries

The ABAP is a template for generating the SQL, but if the variables do not have values assigned at runtime (e.g. if i_eipa-infnr is not assigned), then they are not included in generating the SQL.

ABAP FOR ALL ENTRIES uses an internal table with a list of items to be selected. The internal table restricts the rows to be retrieved. If the internal table is empty, no selection conditions (local predicates in the SQL) will be created for the columns referencing the internal table.

Here, the ABAP uses “FOR ALL ENTRIES”, but when the internal table used in the FOR ALL ENTRIES is empty, then no local predicates on infnr, ekorg, or werks are created, so the only local predicate in the SQL is the MANDT = that is automatically added by ABAP.

The program is custom code (Zxxx) and so we ask the programmers to review the ABAP, in order to modify it to check at runtime whether the internal table i_eine is empty, and if so to skip the call.

7.4.3.1. Summary of Performance problem – empty itab with For All Entries

The majority of time in ZPUR11 is DB time, so we use ST05 trace. We find that calls to EINE are over half of DB time, and that the SQL has a local predicate only on MANDT column, which is unusual. When we display the ABAP, we find that there is a FOR ALL ENTRIES select using an internal table. We know that when the internal table is empty, then SAP leaves out the local predicates for the columns referencing the internal table. In this case that leaves no selection conditions in the ABAP, so the entire table was selected. We return the program to the developers to check the condition that the internal table is empty.

An issue that looked at first glance like a DB performance problem, since DB time was over 90% of transaction time, is an issue in the ABAP logic.
7.4.4. Performance problem caused by data skew

We’ve been asked to review the performance of RLLL01SE, a periodic batch job that runs frequently, and where about 99% of the response time is DB time. This is a symptom of a likely problem related to DB calls.

We trace the statement with ST05, and see that it ran nearly 5 minutes (297 seconds) to retrieve 18 rows. This is very slow.

Figure 43: RLLL01SE long DB time

Extend SQL Trace List

Figure 44: RLLL01SE trace

Explain the statement as in Figure 45, and see that DB2 is using HSJOIN as the access method. This is a little unusual, given that there are only local predicates on LTAP. SAP transactional SQL is usually very simple, and in general when two tables are joined, the filtering local predicates are on one table, so DB2 uses NLJOIN to access that table first, then joining to the inner table. Why does DB2 not use LTAP as the outer table with NLJOIN?

HSJOIN is what DB2 will choose when it expects that it will retrieve many rows from both tables, or if there is not a good index to the inner table for NLJOIN. Let us look at the estimated rows in the explain, to see whether DB2 might be choosing this access path because it expects many rows to be retrieved.
In Figure 45 note that DB2 estimates that 64,000 rows will be returned. But we know from the SQL trace in Figure 44 that DB2 actually returned 18 rows. This looks like a case where DB2 was not able to make a good estimate of filtering, and so chose a bad access path.

Drill into the blue LTAP in Figure 45, to see how large the table is, and to see the indexes.

Figure 46: LTAP has 12M rows
Figure 47: LTAP cardinality of local predicate columns

Figure 47 shows that all three columns with local predicates in the SQL have low cardinality (MANDT = 1, PQUIT = 2, WERKS = 11). With only cardinality information, DB2 assumes that each combination of MANDT, PQUIT, WERKS will return \((12.9M / (1 \times 2 \times 11)\) rows, that is 586K rows. So, DB2 optimizer believes it will be faster to make one pass merging the joined tables together with HSJOIN, rather than accessing LTAP first, and joining each row that qualifies in LTAP to LTAK with NLJOIN.

We can also see DB2's estimate of filtering by using Details on LTAP–M in Figure 45. DB2 thinks that PQUIT = ? will select one half of the rows (filter factor is 5 * 10^{-1}).

Figure 48: DB2 filtering estimate for PQUIT

By default, SAP prepares with parameter markers (not values) and so without ABAP hints DB2 can use only column cardinality when estimating filtering. We need to check the value used at runtime, and look at the distribution of values, to see if the distribution is skewed.
When we display the source code in Figure 49, we see that it is SAP code, and that PQUIT is always set to “space”. We can also get the values used at execution time from the ST05 trace in Figure 44.

If there is data skew in the columns with few distinct values, and if the program is retrieving infrequently occurring values, it may be better to use NLJOIN. Let’s find out what the PQUIT column means, and determine if it is skewed. In the SE16 in Figure 50, place the cursor in the PQUIT field, and press F4 to list the possible values. There are two, which stand for yes (complete) and no. It is likely that most rows are complete. Note also that space is the “no” value, so the literal con_pquit_space in the ABAP in Figure 49 is likely to be always retrieving few rows. We will count the rows below.

In Figure 51, we use SE16 to count rows. Specify selecting rows with “space”, and press “Number of Entries” to count the rows with PQUIT = “ “ (space).
So, we see that there are very few rows with PQUIT=space (7K of 12M table rows). Thus, using NLJOIN to access LTAP first would probably be better than using HSJOIN. With 11 values of WERKS, on the average there would be about 700 rows with PQUIT=space per WERKS.

It is also possible to use “SQL commands” in DBACOCKPIT to get row counts.

This is SAP code and SAP tables, and normally we would go straight to Service Marketplace after identifying the program, rather than taking the detour above to find out where the filtering is. But
if the code were custom code, then we would have had to analyze whether there was skew causing a slow access path.

Skew problems can be solved either by using ABAP hints together with DB2 histogram and frequency statistics, as described in SAP notes 129385 and 150037, or by using “Optimization Guidelines” which are described in note 868888.

Once we gather DB2 statistics, if we explain the statement using the literal space (‘ ’) in the SQL, DB2 can now give a better estimate of the filtering.

<table>
<thead>
<tr>
<th>relational operator:</th>
<th>equal (=)</th>
</tr>
</thead>
<tbody>
<tr>
<td>subquery input required:</td>
<td>No</td>
</tr>
<tr>
<td>filter factor:</td>
<td>9.9999E-03</td>
</tr>
<tr>
<td>predicate text:</td>
<td>Q1.PQUIT = ' '</td>
</tr>
</tbody>
</table>

**Figure 54: PQUIT filtering estimate improved**

**ABAP Hints such as “&SUBSTITUTE VALUES&” are suitable when the optimal access path could change depending on the program inputs, for example when the SQL might choose a large percentage or small percentage of rows depending on the inputs from the user.**

**Optimization guidelines are suitable when one knows what the best access path is ahead of time, and can usually be used in place of the “&SUBSTITUTE LITERALS&” ABAP hint.**
We search SMP, and find a SAP note 908363 that fixes the problem by using an “optimization guideline” to specify NLJOIN.

![Figure 55: SAP note 908363 correction](https://websmp2.330.sap-ag.de - Display source for correction - Microsoft Internet Explorer)

### 7.4.4.1. Summary of Performance problem caused by data skew

ST03N shows DB time is large percent of elapsed time, and ST05 trace shows long running SQL that retrieves few rows. Explain shows HSJOIN, though all local predicates are on one of the two tables. There is a literal in the SQL for one column in the ABAP, and the literal is an infrequently occurring value in a low cardinality column. Since the default SAP prepare method in ABAP does not pass values, DB2 can only use cardinality (number of distinct values) to estimate filtering -- DB2 cannot tell that an infrequently occurring value is sought. SAP provides a note that fixes the problem with an Optimizer Guideline, to specify the join type and index.
7.4.5. Performance problem - table buffering candidate

We’ve been asked to review the performance of an SAP program which has long DB times. We take an ST05 trace and summarize it.

![Summarized SQL Statements](image)

**Figure 56: ZBC1_BC_PARMS ST05 summary**

The table with the second longest duration is a custom table (starts with Z). The BfTp is “cust”. If we place the cursor on that and press F1, SAP displays the meaning of the BfTp values.

![Table buffer type](image)

**Table buffer type**

If no buffering is allowed, space is displayed.

If buffering is allowed and activated, the following are displayed as the buffering type:

- full: Fully buffered
- gap: Gently buffered
- snap: Individual records buffered

If buffering is allowed but deactivated, the system displays the following as the buffering type:

- default: Full buffering possible
degap: Generic buffering possible
desnap: Individual record buffering possible
default: No buffer type specified

If the table is of type “APPL2” - organization and Customizing - and is not buffered, the ID “cust” is displayed.

**Figure 57: buffered table type**

The table is type “cust”, which means it is customizing but not buffered. Customizing data is frequently buffered in SAP, and since it is a Z table, the technical settings may not have been set correctly when the table was created. We use ST10 to look at the statistics on whether the table is changed frequently, since buffered tables should not be frequently changed.
Since the table is not buffered as we saw in Figure 56, in ST10 we select “not buffered” for “all servers”. Order the report by calls.

ZBCI_BC_PARMS has no changes over the week. Next, check the size of the table. We have seen previously in Figure 40 that when a statement is explained, one can drill in on the table to see the RUNSTATS data such as cardinality (i.e. the number of distinct values). This information can also be retrieved using DB02 transaction “single table analysis”
Figure 60: DB02 ZBCI_BC_PARMS

DB02 shows that at the time of the last RUNSTATS, there were 75 rows in the table.

The third test for buffering is whether the application can tolerate a short period where the data on the app servers is out of synch, when changes are made. We would need to contact the functional team to determine whether this table passes the third test. If so, then the technical settings could be changed in SE13 to fully buffer this table.
7.4.5.1. Summary of Performance problem - table buffering candidate
Summarized ST05 trace showed a frequently accessed Z table that was not buffered. ST10 showed that the table was infrequently changed. DB02 showed the table was small. We discuss buffering the table with the application team.

This is another example of how we can reduce DB call time by making changes in SAP, rather than making changes to the database configuration.

7.4.6. Performance problem where predicates do not match indexes
In this example, we have been asked to review the performance of a batch job. In the SQL traces, we find that there are slow selects – over 250 ms to retrieve no rows.
Explain the statement – there are local predicates on MANDT, FKART, VBELN, and KNUMV. The local predicates only match one column (MANDT) on the index, so the entire index is scanned.

```
SELECT *
FROM
"/IRM/IPSIHDR"
WHERE
"MANDT" = ? AND "FKART" = ? AND "VBELN" = ? AND "KNUMV" = ?
FETCH FIRST 1 ROWS ONLY
```

Figure 63: /IRM/IPSIHDR explain

When we review the indexes, we see that if the SQL contained “IPTYP=” in the WHERE clause, it would match three columns in the index. Since VBELN has many distinct values, this should be an efficient index access.

```
+-----+--------+---------+
| Name | Distinct |
| MANDT | 1       |
| IPTYP | 7       |
| VBELN | 282,624 |
```

Figure 64: /IRM/IPSIHDR–001 index

We need to check the SQL, to see whether the problem is that the ABAP does not have IPTYP, or whether it is in the ABAP but not set at runtime.

```
select single * from /irm/ipsihdr
into wa_hdr
where FKART = vbrk-FKART
and VBELN = vbrk-VBELN
and ENUM = vbrk-ENUM.
```

Figure 65: /IRM/IPSIHDR ABAP
In Figure 65, we find the SQL is in an ABAP enhancement, and that it does not contain IPTYP in the where clause. At this time, we follow our decision matrix in Section 8.1.6 – ask the ABAP programmer to change the program to better use the index. There are few values of IPTYP, so if the value is not known, it might also be possible to add IPTYP IN to the ABAP, and specify all possible values. This would be more efficient than scanning the entire index, as is currently done.

7.4.6.1. Summary of performance problem where predicates do not match index

SQL trace shows long calls retrieving few rows. Explain shows that local predicates match only one column in the index, so index is scanned. Review the ABAP and compare it to the index, and we see that if IPTYP can be added to the WHERE clause, we would match three columns in the index, and the access would be efficient.

7.4.7. Runtime analysis of SQL when you cannot use ST05

ST05 SQL trace is easy to use and comprehensively integrates DB2 information (e.g. Explain, RUNSTATS, Tables and Indexes) and SAP information (e.g. call time and runtime parameters) together for analysis of SQL. However, it has one feature that prevents it from being used for analyzing all slow SQL problems – one must trace both the start and end of a database call with ST05. If there is very long running SQL, and we are contacted in the middle of the job (after the SQL starts running) to analyze what is going on, we cannot use ST05. In this case, we can use ST04 to display the DB2 applications, and explain the SQL. Here, we have been asked to evaluate why ZFIAP_INVOICE is running so slowly. In SM50, we see that it is constantly accessing the same table, BKPF. We start an SQL trace on it. After a few minutes, when we try to process the trace, there is no trace data.

Take note of the app server and PID where the program is running.
Run ST04, and select Applications. Filter the report (the funnel in Figure 67) for active threads only.

![Figure 67: ST04 applications](image)

Find the row for the PID of the program in Figure 67, select the row, and press Details (magnifying glass).
Then select the “Statement Text” tab shown in Figure 68, to see the SQL.

![Performance: Application Snapshot](image)

**Figure 68: ST04 applications statement text**

This structure of SQL (with all the CASTS) is how SAP processes a FOR ALL ENTRIES by building a temp table with the keys for rows to be retrieved, and joining the temp table to the table from which the rows are selected. (One would press the “Source” button in Figure 68 to see the ABAP). The ABAP source is shown in Figure 71.
If we explain it, we get our first look at why this runs so slowly – there are local predicates on MANDT, BVORG, BLART, and BSTAT, but MANDT = is the only local predicate that matches the index (see “#key columns” in Figure 69). Since the second column of the BKPF~ZDF index is BUKRS, which has no local predicate, the entire index is scanned to process the local predicates on BVORG, BLART, and BSTAT. Moreover, since every row in the temp table with the key values is joined to the BKPF table (NLJOIN is used), the index is scanned once for each value that is sought.

Figure 69: ST04 applications explain

In order to confirm that the local predicates on BLART, BVORG, and BSTAT are processed Sargable on the index (though not index matching), select the index name (BKPF~ZDF) and press the “Details” button.
Figure 70: Optimizer Details

In Figure 70 note that MANDT is the only column used as a start and stop condition to specify the range of the index search (index matching access). The other local predicates BSTAT, BLART, BVORG are Sargable – they are processed as each index entry is read.

We display the ABAP by pressing the Source button in Figure 69.

Figure 71: ST04 applications source
This is custom code accessing a standard table, for which a custom index (BKPF~ZDF) has already been created. We would review the SAP notes listed in Section 8.1.6 to determine if this is an error in using the SAP data model, then we would send the program to the ABAP and DBA teams, in order to review whether the program can be modified to match the available indexes, or whether possibly the BKPF~ZDF index could be modified by changing the order of the columns, so that BKPF~ZDF supported this SQL, as well as the SQL for which it was designed.

7.4.8. Invalid parameters cause unnecessary SQL

Logic errors, such as not validating input parameters or the results of SQL can create the situation where SQL is executed with parameters that will never retrieve a row, since the parameters specify keys that cannot exist. The SQL cache does not usually have parameter values, so this type of problem is generally found when analyzing an SQL trace.

In the following example, we have taken an SQL trace of a long-running program. Note that the value for MATNR passed from the ABAP is ‘ ’ (space), and these selections never retrieve any rows – there is no MATNR with the value of space.

We display the ABAP source for the MARA select, and this is a Z program.
As shown in Figure 74, format the trace in Figure 72 for identical selects.

We see in Figure 75 that the selects on MARA are often executed for MATNR = ' ' (space). This appears to be a logic error in the program that needs to be investigated.

We return the program to the ABAP development team, so they can determine why MATNR has no value on some of the executions.

7.5. **Majority of time is not CPU or DB request time**

There are a number of SAP response time categories that can also indicate performance problems that are not covered in this paper. SQL and ABAP, which are covered in this paper, are the source of the vast majority of performance problems.

The paper ‘Tuning SAP on DB2 for z/OS on System z” on IBM Techdocs (http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100287) contains detailed explanations about additional SAP sources of program delays (such as ENQ delay, SAP sort, local I/O, commit work and wait, RFC, GUI, network, etc), their symptoms and possible fixes.

8. **System performance problems**

8.1. **Perform SQL Cache Analysis**

In order to find inefficient SQL that may have a system-wide impact, because of using excessive CPU or doing excessive I/O, review the SQL cache via ST04.

8.1.1. **Indicators of inefficient SQL access – Netweaver 7.10**

With SAP Netweaver 7.10, there are statement level “efficiency metrics” which can be used to find statements in the SQL cache that are inefficient, that is to find statements that have to search many rows or pages of data to return the result. As discussed above, a statement can be executed...
inefficiently for a number of reasons – misuse of the SAP data model, incorrect or out of date RUNSTATS, or programming errors, or missing indexes. Here we will show how to use the SQL cache to find candidates to examine.

The three key columns are:

- **BP Gets / Executions** – how many pages (data and index) are referenced each execution. When this is high, it is not necessarily a problem. If an SQL statement returns many rows, then it is normal to have many BP Gets per Execution.
- **BP Gets / Rows Processed** – how many pages are referenced for each row returned to the ABAP program. In general, when this is high, it is a symptom of inefficient SQL. There is an exception which is shown in Figure 79.
- **Rows Examined / Rows Processed** – how many table rows are examined for each row returned to the ABAP program. When this is high, it means that local predicates (where column op value clause) are being evaluated on the table, not in an index. In general, when this is high it is a symptom of inefficient SQL. An exception is shown in Figure 79.

A statement that selects a row at a time from the DB, and where all the local predicates can be efficiently processed in the index (e.g. all the local predicates are equals predicates) might perform 3-5 BP Gets / Row Processed. An array fetch that retrieves many rows per execution and also uses indexes efficiently might perform less than one BP Get per Row Processed.

With a statement that performs 10-20 BP Gets / Row Processed (or higher) there may be an opportunity to improve the efficiency (and speed) of the statement. The opportunity to speed up the statement is related to how inefficient it is – if a statement performs 1000 BP Gets / Row Processed, then it might be possible to reduce the elapsed time by 95% or more. If the statement performs 10 BP Gets / Row Processed, you might only be able to reduce the statement execution times by a few percent.

![Diagram](image)

**Figure 76: Inefficient SQL in ST04 cache**

In the example in Figure 76, each statement runs about 90 ms, but BP Gets / Execution is high (indicating that many pages were referenced) and BP Gets / Row Processed is high, which indicates that many pages were searched in order to return few rows. Together, these indicate that the statement is processed inefficiently. In addition, Rows Read / Rows Processed is high, which indicates that the local predicates are being evaluated on the table, rather than in an index.
In Figure 77, the statement takes on average over 8 seconds to execute, and performs about 15K BP Gets / Execution, and about 15K BP Gets per Row Processed. The statement is executed inefficiently. However, the Rows Read / Rows Processed is low, which indicates that local predicates are not being evaluated on the table rows. When we explain the statement, we see how this happens in Figure 78.

In Figure 78 note that DB2 is using index DD03L~5, but “#key columns” is zero. The index contains one of the local predicate columns, but DB2 cannot do index matching access, since there is no local predicate on the first column in the index (TABNAME). As always when there is SQL where the predicates do not match the indexes, one would evaluate whether the program can be changed, or the user inputs to the program can be changed, and whether the problem has a large impact that needs to be fixed, before creating a new index on the table.

In Figure 79 we see why we need to use BP Gets / Execution together with BP Gets / Row Processed. In this example BP Gets / Row Processed is high, which would normally indicate an inefficient statement. But BP Gets / Execution is low, which indicates that each execution
references just a few pages. The “Rows Returned” column is the key to understanding this. If a statement seldom or never returns a row to the application, then BP Gets / Row Processed will be high, even if the statement is executed efficiently in DB2.

In the case where a statement never returns a row to the application, the ABAP should be evaluated by application experts to determine whether the select is really needed. Or, if the table is a table where buffering is allowed, and the table is empty, then buffering the table in SAP SE13 will allow the ABAP to determine that no rows exist without calling the DB server.

**8.1.2. Indicators of inefficient SQL access – Netweaver 7.0 and earlier**

In earlier SAP versions there are not “yield indicators” or “efficiency metrics” at the statement level. That is, one cannot compare “Logical Reads” or BP Gets with rows returned to the application, in order to calculate how many pages were referenced per row. This is available with Netweaver 7.10.

Lacking efficiency metrics on the statements, we can search the SQL cache for SQL statements that have high total elapsed time, or high total rows read in the table, and then examine the SQL to see if there may be inefficient processing that is causing these symptoms.

When you sort the SQL cache by “Total Execution Time”, you can review statements that have the most I/O activity, and cause the most delay in the system.

When you sort the SQL cache by “Total User CPU” or “Rows Read”, you can search for statements that have inefficient access paths, where the data pages referenced are frequently in memory.

![Figure 80: ST04 ordered by rows read](image)

Select the first statement in Figure 80, and press ‘Explain’.

© Copyright IBM Corporation, 2012
In Figure 81, note that the entire table is read on each execution – DB2 uses TBSCAN.

In order to see the table statistics, and the indexes on the table, double click in Figure 81 on the blue BSIP next to TBSCAN.

Now, by comparing the local predicates (the column operator value clause) with the index, we can see that DB2 has chosen to scan the table, since the local predicates match MANDT, but there is no local predicate on the second column in the index, BUKRS. Rather than scan the index and then read the table, DB2 chooses to scan the table.

It is important to note that at this point, we don’t know how many rows are returned by these calls to BSIP. With NW70, we need to trace the execution of the statement to get the number of rows returned. For instance, while the statement in Figure 82 has an equals predicate on XBLNR, and
XBLNR has 2.9M distinct values, there are 3.7M rows in the table, so on the average, there would be just over one row per XBLNR. However, it is possible that there could be a single value of XBLNR (e.g. space) with 800K rows, while every other value of XBLNR has a single row.

Figure 83: BSIP ST05 trace

With the SQL trace in Figure 83 we can determine how many rows are returned. See “Recs” in the FETCH call in Figure 83. No rows are returned in this example. This information is useful when evaluating whether a new index is needed to optimize performance. If this statement returned a million rows on each execution, then TBSCAN would probably be the best way to process it. However, since the calls frequently return no rows, and one second for zero rows is very slow, we would go to our decision matrix in Section 8.1.6 to determine whether to modify the ABAP, or create an index, or search SAP Service Marketplace.

8.1.3. Index-Only Access

Following is another example of SQL that accesses the database inefficiently. In Figure 84, we have a statement with high average execution time (over three minutes) but only 27 rows read per execution (1898/69). How can this be?
Figure 84: high elapsed time with few rows read

When we explain the statement as shown in Figure 85, we can see what is happening. Only one column in the BKPF~ZDF index (MANDT) is matched. The rest of the local predicates (BUKRS <> BLART <>, BVORG =, and BSTAT =) can be processed in the index, so the entire index with millions of entries is read, but only a few table rows are retrieved. As we saw in Figure 70, we could use the “Details” button in explain to confirm whether the local predicates are processed on the index or table.

A note about <> (not equals) – it is never index matching. If possible, it is always more efficient to specify the values being retrieved, rather than values to not be retrieved. Using “column = value” or “column IN (value1, value2, etc)” is always more efficient than “column <> value3”.

© Copyright IBM Corporation, 2012
This next example shows how “rows read” are table rows read, and that a statement can still be slow and inefficient, even if few rows are read per execution. For example, if the SQL can be processed using index only access, where only columns in an index are referenced, then “Rows read” will be zero.

In Figure 86, note that there is a statement that runs about two seconds per execution, but reads no rows.

When we explain the statement as shown in Figure 87, we see why there are no rows read – DB2 uses index only access – IXSCAN and RETURN – matching MANDT in the index, and screening the other local predicates in the index. At this point we cannot tell if the statement is inefficient or not, as we do not know the number of rows returned per execution – if all the columns in the SQL
are in an index, DB2 can return a row to the application without reading a row from the table. We
would need to trace with ST05 to determine if few rows are returned, or many are returned.

Figure 87: BSIP index only access

If you compare the BSIP statement in Figure 87 with the BSIP statement in Figure 82, you will
find that this statement references no columns outside the BSIP~0 index, while the other statement
in Figure 82 references the SHKZ6 column, and so in the example in Figure 82, DB2 must read the
table row after accessing the index.

8.1.4. Performance problem - local predicates do not match indexes

In this example, we are ordering the cache by rows read, in order to search for statements that do
not use indexes effectively. We choose a statement on VLCVEHICLE which takes over 900 ms
per execution, and reads over two hundred thousand (355 million rows / 1462 executions) rows per
execution. We cannot tell how many rows are returned to the application.
Select the statement to be examined, and press ‘EXPLAIN”. Then drill in on the blue “VLCVEHICLE” text in the explain to see the table and its indexes.

The local predicates in the SQL are MANDT = and VHCEX =. MANDT is in two indexes, but there is only a single value, so it is unlikely that it filters well – there is only one productive MANDT in this system, so all business applications would run in that MANDT. We need to check how many distinct values there are of VHCEX, to see if an index would be likely to improve this. (If there is only one value of VHCEX, and we are always selecting that value, then we would retrieve the entire table, and using an index would not help.)
We can see how many distinct values there are of VHCEX by pressing “Columns” in Figure 89, but we cannot see if the distribution of data is skewed (some values are common, some uncommon). One way to find out both the number of distinct values and the skew is using DB05. If you need to determine the cardinality of a large table, do not use DB05 – do the counts with DB2 commands.

Figure 90: DB05

In Figure 90, we run DB05 for the column VHCEX in VLCVEHICLE.

Figure 91: VLCVEHICLE DB05 results

In Figure 91, we see that there are 243,150 rows, with 33,117 distinct values. The data distribution is very even – every distinct value has between 1-10 rows in the table. Since the SQL has an
equals predicate on VHCEX, then each execution of the SQL is returning at most 10 rows to the application, but it is currently taking over 900 ms to do this. This is very slow. A 10 row fetch using an index should take just a few ms, at most. So, an index on VHCEX should help to improve the performance.

SAP delivers some indexes defined in the data dictionary, but not active in the database. We can use SE11 to check the DDIC, and see if this is such a case.

**Figure 92: SE11 VLCVEHICLE**

Then press the “Indexes” button.

**Figure 93: SE11 VLCVEHICLE indexes**

The primary index (VLCVEHICLE~0) is not displayed in this list, and there are three indexes here. In Figure 89 we saw a total of three indexes, so there must be one that is defined in the DDIC that is not active on the database.

Drill into the EXT index in Figure 93, and we see that there is an index (MANDT, VHCEX) delivered in the DDIC, but not active on the DB. SAP does this with optional indexes.

© Copyright IBM Corporation, 2012
Figure 94: VLCVEHICLE~EXT index

So, we can activate the optional VLCVEHICLE~EXT SAP index in the DB (e.g. using SE14).
In more recent versions of SAP, the SE11 index list contains a column that shows if the index is defined or not. The “O” in the “DB status” column means the index does not exist on the DB.
8.1.4.1. Summary of Performance problem - local predicates do not match indexes

We found high impact SQL statement which read many rows in the DB. When we explain the SQL, we see that the indexes on the table do not support the local predicates in the SQL. We check cardinality and skew of the column VHCEX, and it has high cardinality and would be a good candidate for an index. Checking SE11, we find there is an optional SAP index (in the DDIC but not active on the DB) that matches this SQL, so we can activate the index to improve the performance.

8.1.5. Reasons that the predicates do not match indexes

There are three basic scenarios:

- The program is not using the SAP data model correctly, or
- The programmer left out selection conditions on the SQL that could have been included,
- The business process may require a new index on the table

Many kinds of information are redundantly stored in SAP. For instance, the transaction data for a billing document may contain columns with the document number for the delivery or sales order being billed. But just because the information is present does not mean that is the right way to retrieve it. Billing documents, to continue the example above, are indexed to support lookup by billing document number. If a program tries to access the billing tables using the delivery column, in order to find the associated billing document, the column with the delivery number will not be indexed, and access will be slow. **If this is the situation, the solution is not to create an index, but to use the SAP data model correctly.**

There is an example of incorrect use of the SAP data model below in section 8.1.8.

SAP has several SAP notes that describe wrong and right ways to use the SAP data model to retrieve application information:

- MM – SAP note 191492
- SD – SAP note 185530
- PP – SAP note 187906

8.1.6. Actions when local predicates do not match indexes

Sometimes, a company’s business processes require new indexes on tables. Here are some guidelines on how to approach the problem when you find a program where the SQL predicates do not match the indexes:

<table>
<thead>
<tr>
<th>ABAP Creator</th>
<th>Table Creator</th>
<th>Action when predicates do not match indexes</th>
</tr>
</thead>
</table>
| SAP          | SAP           | • Check data dictionary for SAP indexes which are not active in DB  
|              |               | • Check SAP notes  
|              |               | • Open message to SAP |
Rewrite program if possible so that ABAP matches available indexes
Evaluate table access patterns, whether table can be buffered
Create index

Check data dictionary for standard indexes that are not active in DB
Review use of data model (will often fix problem)
Change program
Create index (should seldom be necessary)

8.1.7. DB2 Index Advisor in ST04

The Index Advisor is available from ST04 SQL Cache. It is an expert system that can offer advice on how to create new indexes. However, it cannot do several important things that need to be done before an index is created:

- It cannot tell if the SAP data model is used correctly
- It cannot tell if the ABAP can be re-written to match the available indexes
- It cannot tell if there is a logic error in the program (e.g. the empty internal table with For All Entries) that affected local predicates in the SQL.

The index advisor also will try to optimize the indexes to the individual SQL statement (for example by including selected columns to give index only access) rather than adding only the most important columns to the index to create a more generally useful index.

Use the Index advisor only after a thorough review of the ABAP, and after a determination that a new index is the best way to solve the problem.

Figure 97: ST04 Index Advisor
Here is an example of using the index advisor to create a virtual index. With this simple SQL, we wouldn’t need to use the index advisor to figure out which columns need to be in the index, but our goal is to show the process to create a virtual index and test its impact on the DB2 access path.

```sql
SELECT T_00."VRKME", T_00."ARKTX", T_00."PATNR", T_00."VBELN", T_00."VKORG_AUFT", T_01."AUART", T_01."VBELN", T_02."ERDAT", T_02."KUNA6", T_02."KUNRG", T_02."VBELN", T_03."KUNNR", T_03."VBELN"
FROM "VBRP" T_00
   INNER JOIN "VBAK" T_01 ON T_01."MANDT" = ?
   AND T_01."VBELN" = T_00."AUEIL"
   INNER JOIN "VBRK" T_02 ON T_02."MANDT" = ?
   AND T_02."VBELN" = T_00."VBELN"
   INNER JOIN "LIK" T_03 ON T_03."MANDT" = ?
   AND T_03."VBELN" = T_00."VESEL"
WHERE T_00."MANDT" = ? AND T_02."ERDAT" = ? WITH UR
```

Figure 98: Sample SQL for index advisor

Other than “MANDT =” (which does not filter), there is only one local predicate in this SQL, so we would expect that DB2 will access VBRK (T_02) first when we explain the statement.

![Figure 99: Explain VBRK join](image)

And as expected, DB2 does. But, the entire VBRK table is read (TBSCAN). When we display the indexes on VBRK, we see that there are no indexes containing ERDAT.

© Copyright IBM Corporation, 2012
Check the Details, to see the DB2 estimate for filtering on the “ERDAT = ?” local predicate.

It filters well, so a new index should help improve performance. Since the local predicate is not on a document number, we may have a situation where we need a new index to enable a new business process “reporting by creation date” on VBRK.

We copy the SQL statement into the index advisor in DBACOCKPIT.
Figure 102: Index advisor

Enter the Schema and Table, and then press the circled button to move the table columns into the column list. Next, add the columns to the virtual index using the arrow buttons.

Figure 103: Index advisor add virtual index

After moving all desired columns to the virtual index, press “Add” to add the virtual index.
Now that the virtual index has been added, we can check that DB2 will use the index by selecting the option “existing, recommended, and user defined indexes”, and pressing EXPLAIN in Figure 104.

The index is used, but Figure 105 shows “key columns: 0” on the virtual index.
But, when we check Details, we see that DB2 would use MANDT = and ERDAT = as index matching (start/stop) local predicates.

<table>
<thead>
<tr>
<th>Predicates:</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 ) Used as Start Condition</td>
</tr>
<tr>
<td>relational operator: equal</td>
</tr>
<tr>
<td>subquery input required: No</td>
</tr>
<tr>
<td>filter factor: 1.00</td>
</tr>
<tr>
<td>predicate text: (C3.MANDT = :?)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8 ) Used as Start Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>relational operator: equal</td>
</tr>
<tr>
<td>subquery input required: No</td>
</tr>
<tr>
<td>filter factor: 1.12</td>
</tr>
<tr>
<td>predicate text: (C3.ERDAT = :?)</td>
</tr>
</tbody>
</table>

Figure 106: Local predicates on virtual index

8.1.7.1. Why one should be cautious with index advisor

If we take the statement in Figure 98, and use “Recommend indexes” in Index advisor, we receive recommendations that are optimized for this statement. As seen above, with a single index containing (MANDT, ERDAT) we can improve the statement, and we would be creating an index that could be used by any business process selecting billing documents (VBRK) by creation date (ERDAT). The index advisor proposes several new indexes which would probably not be suitable for other business processes.

Use index advisor to get ideas from DB2 about how to optimize the statement, and then simplify the indexes (by removing columns) to make them useful for more general business processes.

Figure 107: Index advisor optimization of indexes

8.1.8. Performance problem - incorrect use of SAP data model

We are reviewing the SQL cache or high impact statements, and near the top is a statement that is run very frequently, with each execution taking about ½ second.
We explain the highlighted statement in Figure 108, to see what is being executed.

We see in Figure 109 that DB2 is reading the entire table (TBSCAN). There are local predicates on MANDT and VBELN. We drill into VEPO to display the table and its indexes, to determine why DB2 does not use an index.
Figure 110: VEPO table and indexes

There are two local predicates in the statement in Figure 109, MANDT = and VBELN =. Checking the indexes in Figure 110, MANDT is in both the indexes on VEPO, but VBELN is not in either. Since MANDT has only one distinct value, it is unlikely it will filter rows (all business programs run in the productive MANDT), so DB2 chooses to scan the table. Since this is a standard SAP table, we need to do some additional checking before creating an index.

In Figure 109, press the “Source Code” button to display the ABAP.

Figure 111: VEPO source code is custom (Y or Z) program

The program is a custom program (starting with Y or Z) and the table is a standard SAP table, so per our decision matrix in Section 8.1.6 we need to check if this is a problem where the data model is not used correctly.
As described in Section 7.4.1.2, SAP data is often denormalized, and there is a correct (indexed) and incorrect (not indexed) way to retrieve it. We check the SAP notes in this section, and find that SAP note 185530 describes this problem, and proposes a solution.

<table>
<thead>
<tr>
<th>List Display SAP Note 0000185530</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Note</td>
</tr>
<tr>
<td>b) Search for shipping unit item with delivery</td>
</tr>
<tr>
<td>incorrect:</td>
</tr>
<tr>
<td>SELECT FROM vepo WHERE vttyp = 'J'</td>
</tr>
<tr>
<td>AND vbeln = l_lipo-vbeln</td>
</tr>
<tr>
<td>Correct:</td>
</tr>
<tr>
<td>SELECT FROM vbfa WHERE vttyp_n = 'X'</td>
</tr>
<tr>
<td>AND vbeln = l_lipo-vbeln</td>
</tr>
<tr>
<td>SELECT FROM vepe WHERE venus = vbfa-vbeln</td>
</tr>
<tr>
<td>Note is release-independent</td>
</tr>
</tbody>
</table>

Figure 112: VEPO SAP note 185530

We now send the ABAP to the development team, and ask that they review note 185530 regarding the correct way to use the delivery number to access VEPO (via VBFA) and modify the program accordingly.

8.1.8.1. Summary of Performance problem - incorrect use of SAP data model

ST04 SQL cache has frequently executed statement that is high in total execution time. We explain the statement, and find that DB2 is using TBSCAN to read the entire table, since the indexes do not support the local predicates. The program is custom and the table is standard, so we first investigate whether it is a problem in mis-use of the SAP data model. SAP note 185530 points out the correct way to access the data using the model and standard indexes.

Whenever you see inefficient SQL that accesses a table using a document number and there is no index on the document number column, the root cause is almost always misuse of the data model. Don’t create an index until researching how to re-write the SQL.

8.1.9. Performance problem - Indexes do not support join predicates

Performance problems can also occur when the indexes do not support the join predicates in the SQL. As with the examples previously shown, the root cause may be that the data model is not used correctly, or that the ABAP needs to be changed to add additional selection conditions to match the available indexes (since both join and local predicates are used for index access when joining), or that new indexes are needed.

© Copyright IBM Corporation, 2012
In this example, we have a long running SQL statement that needs to be analyzed. In Figure 113, note that the only local predicate (other than MANDT =) is on LIKP – “T_01.VBELN = ?”. DB2 can use join conditions to transitive apply local predicates on other tables.

```sql
SELECT T_00."VBELN", T_00."VKORG", T_00."KUNNR", T_00."KUNDE", T_00."PARVW", T_00."KUNAG", T_01."ERDAT", T_01."WADAT_IST", T_01."VBELN", T_02."POST_CODE1", T_03."MATNR", T_03."LFIMG", T_03."VRKME", T_03."FOSNR", T_03."VBELN", T_04."VBELN", T_04."BSTAK"
FROM "VLPAA" T_00
INNER JOIN "LIKPP" T_01 ON T_01."MANDT" = ?
AND T_01."VBELN" = T_00."VBELN"
INNER JOIN "ADRCC" T_02 ON T_02."CLIENT" = ?
AND T_02."ADRNR_ME" = T_00."ADRNR_ME"
INNER JOIN "LIPS" T_03 ON T_03."MANDT" = ?
AND T_03."VBELN" = T_01."VBELN"
INNER JOIN "VBAR" T_04 ON T_04."MANDT" = ?
AND T_04."VBELN" = T_03."VBELN"
WHERE T_00."MANDT" = ? AND T_01."VBELN" = ?
WITH UR
```

Figure 113: SQL for join predicate example

Explain the SQL, and review the access path and the row and cost estimates.

Figure 114: Explain for join predicate example
As discussed in Sections 7.2.2 and 7.2.3, the VLKPA~0 IXSCAN step 11 has some symptoms of a problem. The “#key columns” is 1, and since almost all SAP tables start with MANDT, this probably means that the access to VLKPA is not efficient. Likewise, the tot_cost is very high (1.1 * 10^5) on this step, and num_rows (2.2 * 10^9) is low.

Select VLKPA~0 in Figure 114, and press Details to see how the predicates are applied.

Figure 115: VLKPA~0 Details

As expected from the “#key columns: 1” in Figure 114, Figure 115 shows that MANDT is the only index matching predicate, and VBELN is SARGable, so DB2 has to evaluate each index entry to check the value of VBELN. Since MANDT does not filter, the entire VLKPA~0 index is read for each row. VBELN is used in VLKPA because of the join condition T_00.VBELN = T_01.VBELN in Figure 113.

We need to check the indexes on VLKPA. Drill into VLKPA in Figure 114.

Figure 116: VLKPA indexes
There is only one index. It contains VBELN, but as the last column. Let's look at the table.

Select VLKPA in the explain in Figure 114 and press the "Dictionary" button.

**Dictionary: Display Table**

<table>
<thead>
<tr>
<th>Field</th>
<th>Key</th>
<th>Init. Data element</th>
<th>Data Type</th>
<th>Length</th>
<th>Descr...</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MANDT</td>
<td>✔</td>
<td>✔</td>
<td>CHAR</td>
<td>10</td>
<td>0</td>
<td>Client</td>
</tr>
<tr>
<td>KUNDE</td>
<td>✔</td>
<td>✔</td>
<td>CHAR</td>
<td>10</td>
<td>0</td>
<td>Partner number (KUNNR, LIKNR, or PERNR)</td>
</tr>
<tr>
<td>PARTF</td>
<td>✔</td>
<td>✔</td>
<td>CHAR</td>
<td>2</td>
<td>0</td>
<td>Partner Function</td>
</tr>
<tr>
<td>YSTEL</td>
<td>✔</td>
<td>✔</td>
<td>CHAR</td>
<td>4</td>
<td>0</td>
<td>Shipping Point/Receiving Point</td>
</tr>
<tr>
<td>LFLAT</td>
<td>✔</td>
<td>✔</td>
<td>DATS</td>
<td>8</td>
<td>0</td>
<td>Delivery date</td>
</tr>
<tr>
<td>ZECOR</td>
<td>✔</td>
<td>✔</td>
<td>CHAR</td>
<td>4</td>
<td>0</td>
<td>Sales Organization</td>
</tr>
<tr>
<td>LARTT</td>
<td>✔</td>
<td>✔</td>
<td>CHAR</td>
<td>4</td>
<td>0</td>
<td>Delivery Type</td>
</tr>
<tr>
<td>KUNDE</td>
<td>✔</td>
<td>✔</td>
<td>CHAR</td>
<td>10</td>
<td>0</td>
<td>Ship-to party</td>
</tr>
<tr>
<td>KUNAG</td>
<td>✔</td>
<td>✔</td>
<td>CHAR</td>
<td>10</td>
<td>0</td>
<td>Sold-to party</td>
</tr>
<tr>
<td>EPHAM</td>
<td>✔</td>
<td>✔</td>
<td>CHAR</td>
<td>12</td>
<td>0</td>
<td>Name of Person who Created the Object</td>
</tr>
<tr>
<td>VBELN</td>
<td>✔</td>
<td>✔</td>
<td>CHAR</td>
<td>10</td>
<td>0</td>
<td>Delivery</td>
</tr>
<tr>
<td>ADDOR</td>
<td>✔</td>
<td>✔</td>
<td>CHAR</td>
<td>10</td>
<td>0</td>
<td>Address</td>
</tr>
<tr>
<td>ADDOR foreclosure</td>
<td>✔</td>
<td>✔</td>
<td>CHAR</td>
<td>10</td>
<td>0</td>
<td>Address number of the ship-to party</td>
</tr>
</tbody>
</table>

**Figure 117: VLKPA SE11**

This example does not appear in the SAP notes on misuse of the data model (185530, 191492, and 187906). But since this is a document number (VBELN) accessed without a good index, our first thought is that this is a data model usage error. After discussion with the application team, or a bit of googling, we determine that this is a data model problem - VLKPA is a table for deliveries, but indexed for access by partner number. It contains data that is also stored in delivery tables LIKP and LIPS. VBELN can be used for efficient indexed access to LIKP and LIPS (see Figure 118), but VLKPA is designed to be read by KUNDE (see Figure 116).

**Figure 118: LIKP indexed for access by VBELN**

This join and its ABAP will go to the application team, to redesign it to conform to the SAP data model.

© Copyright IBM Corporation, 2012
8.1.10. **Performance problem SAP buffer space**

Here, we are examining the statements in the SQL cache. The cache is ordered by total execution time. The statement on A618 looks OK from the DB2 perspective – average elapsed time is less than one ms.

Figure 119: A618 in statement cache

As seen in Figure 120, the statement is executed efficiently in DB2, matching the first four columns on the primary index. There are ~110K rows in the table, and MATNR has over 20K distinct values, so on the average each select returns about 5 rows.

Figure 120: A618 Explain

However, in SAP tables with names like Annn are pricing condition tables, which are frequently buffered for performance. We need to check why it is not buffered. It could be that the table is too large, or frequently changed and invalidated.

© Copyright IBM Corporation, 2012
First we use SE13 as shown in Figure 121 to check the buffer settings.

![Dictionary: Display Technical Settings](image)

Figure 121: SE13 A618

SE13 shows it is set to be buffered, which is what we expected to find for a condition table.

When condition tables are very large (10s or 100s of MB) then they are often set to be not buffered.
A shown in Figure 122 to Figure 125, we can use ST02 to check the contents of SAP buffers and review why A618 is not actually buffered in SAP memory.

**Tune Summary**

<table>
<thead>
<tr>
<th>Buffer</th>
<th>HitRatio</th>
<th>Alloc KB</th>
<th>Frames KB</th>
<th>% Free Sp</th>
<th>Dir. Size</th>
<th>FreeDirEnt</th>
<th>% Free Dir</th>
<th>Snaps</th>
<th>DB Accs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Buffer</td>
<td>99.96</td>
<td>89,592</td>
<td>1,286</td>
<td>49.45</td>
<td>99,608</td>
<td>44,581</td>
<td>49.45</td>
<td>0</td>
<td>115,227</td>
</tr>
<tr>
<td>Field definition</td>
<td>99.96</td>
<td>100,931</td>
<td>13,952</td>
<td>44.24</td>
<td>98,608</td>
<td>74,315</td>
<td>57.57</td>
<td>0</td>
<td>31,726</td>
</tr>
<tr>
<td>Short HTAB</td>
<td>99.97</td>
<td>5,012</td>
<td>1,197</td>
<td>51.59</td>
<td>22,568</td>
<td>18,099</td>
<td>51.59</td>
<td>0</td>
<td>3,014</td>
</tr>
<tr>
<td>Initial records</td>
<td>73.70</td>
<td>22,013</td>
<td>10,207</td>
<td>51.49</td>
<td>22,508</td>
<td>4,287</td>
<td>19.10</td>
<td>0</td>
<td>18,267</td>
</tr>
<tr>
<td>program</td>
<td>99.96</td>
<td>2,400,000</td>
<td>2,247,242</td>
<td>73.64</td>
<td>1,268,008</td>
<td>1,167,723</td>
<td>87.31</td>
<td>0</td>
<td>103,165</td>
</tr>
<tr>
<td>CUA</td>
<td>99.96</td>
<td>16,000</td>
<td>5,039</td>
<td>49.23</td>
<td>9,600</td>
<td>8,481</td>
<td>91.81</td>
<td>0</td>
<td>4,834</td>
</tr>
<tr>
<td>Screen</td>
<td>99.97</td>
<td>10,531</td>
<td>31</td>
<td>8.17</td>
<td>18,000</td>
<td>9,225</td>
<td>52.25</td>
<td>0</td>
<td>1,368</td>
</tr>
<tr>
<td>Calendar</td>
<td>97.16</td>
<td>486</td>
<td>366</td>
<td>78.57</td>
<td>200</td>
<td>53</td>
<td>28.50</td>
<td>147</td>
<td>0</td>
</tr>
<tr>
<td>STR</td>
<td>99.95</td>
<td>4,096</td>
<td>3,366</td>
<td>99.79</td>
<td>2,000</td>
<td>1,997</td>
<td>99.79</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tables</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Generic Key</td>
<td>99.96</td>
<td>97,956</td>
<td>4,363</td>
<td>9.18</td>
<td>18,008</td>
<td>1,789</td>
<td>17.89</td>
<td>577</td>
<td>20,825,475</td>
</tr>
<tr>
<td>Single record</td>
<td>99.35</td>
<td>60,000</td>
<td>6,277</td>
<td>18.56</td>
<td>560</td>
<td>357</td>
<td>71.40</td>
<td>0</td>
<td>963,748</td>
</tr>
<tr>
<td>Export/Import</td>
<td>99.40</td>
<td>61,440</td>
<td>19,569</td>
<td>56.67</td>
<td>28,000</td>
<td>19,444</td>
<td>52.22</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Exp./Imp. SCR</td>
<td>99.70</td>
<td>4,096</td>
<td>3,163</td>
<td>94.65</td>
<td>2,000</td>
<td>1,993</td>
<td>99.79</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 122: ST02 A618**

In Figure 122 we see that there is only about 4M free memory out of a total of about 97MB in the Generic Key Buffer. There are millions of calls for DB Access on the Generic Key buffer, which has about 30 times as many accesses as the next highest buffer. This is not normal if buffering is effective. Drill in on the “Generic Key” line in Figure 122.

**Tune: Detail Analysis**

<table>
<thead>
<tr>
<th>Efficiency</th>
<th>HITRATIO</th>
<th>%</th>
<th>99</th>
<th>DB access quality</th>
<th>%</th>
<th>98</th>
<th>DB access</th>
<th>%</th>
<th>98</th>
<th>DB access saved</th>
<th>%</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Allocated KB</td>
<td>97,656</td>
<td>Available KB</td>
<td>94,440</td>
<td>Used KB</td>
<td>88,137</td>
<td>Free KB</td>
<td>4,603</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Available KB</td>
<td>18,680</td>
<td>Used</td>
<td>8,281</td>
<td>Free</td>
<td>1,789</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Snaps</td>
<td>Objects swapped</td>
<td>6,818</td>
<td>Frames swapped</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reset</td>
<td>Total</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 123: ST02 detail A618**
Press the “Buffered Objects” button in Figure 123, and sort by calls.

### Figure 124: ST02 buffered objects

In Figure 124, note that the A618 table is in ‘error” state. Click on the word “error” and press F1 for SAP help.

### Buffer status of a table (long form)

1. **Valid**: The table content in the buffer is valid. Mixed access takes place in the buffer.
2. **Invalid**: The table content in the buffer is invalid. Mixed access takes place in the buffer.
3. **Loadable**: The table content can be loaded from the database, and the buffer is empty. Normal access takes place in the database.
4. **Invalidated**: The table content can be invalidated from the buffer and the buffer is empty. Normal access takes place in the database.
5. **Buffered**: The table content can be accessed from the buffer. Normal access takes place in the database.
6. **Overflow**: The table content cannot be adjusted in the buffer. Mixed access takes place in the database.
7. **Invalidated**: The table content cannot be adjusted in the buffer. Mixed access takes place in the database.
8. **Overflow**: The table content cannot be adjusted in the buffer. Normal access takes place in the database.
9. **Invalidated**: The table content cannot be adjusted in the buffer. Normal access takes place in the database.
10. **Invalidated**: The table content cannot be adjusted in the buffer. Normal access takes place in the database.

### Figure 125: ST02 help

Error means that it will not fit into the generic buffer. We need to check how large the table is, in order to determine whether we can increase the generic buffer enough to hold it. We can use DB02 single table analysis.
The table is about 10MB, which is too large to fit into the currently sized Generic Key buffer, but still in the bufferable size range. We can buffer the table by increasing the size of the SAP Generic Key Buffer.

We saw in Figure 124 that the table is very seldom changed, but to reduce the impact of changes, we can further investigate whether it could be generically buffered, rather than fully buffered.

From Figure 120, we know there are 66 distinct values of KSCHL, the third column in the primary index. If the table buffering were changed to generic on 3 index columns, then SAP will buffer the rows for each KSCHL value separately, so that if one row is changed, only the rows with the same KSCHL value will be reloaded, rather than reloading the entire table (for fully buffered table) every time any single row is changed.

There are over 20K values of MATNR, so setting the table to be buffered on four columns would create thousands of generic ranges, which is too many. Figure 122 shows that there are currently 10K directory entries for Generic Key buffer, and each generic range uses a key entry.

8.1.10.1. Summary of symptom of SAP buffer problem

ST04 SQL cache shows that A618 is frequently selected, but SE13 shows it is set to be buffered in SAP. We check the sap instance buffers with ST02 and the size of the table with DB02, and the Generic Key buffer does not have enough space to hold the entire A618 table. We review the data in the table, and determine that the table could be fully buffered (after increasing the size of the Generic Key buffer), or the buffering could be changed to generic on 3 index columns.
### 8.1.11. **Performance problem – missing index on custom table**

When new functionality is added via custom programs and custom tables, it can happen that not all indexes are created to support efficient access. Since most testing is done with small sets of data and low transaction volumes, problems with missing indexes may be found only after the new functions are moved to production and the table grows over time.

In this example, we have ordered the SQL cache by CPU time, to find high impact statements.

![Figure 127: Y-table with high CPU activity](image)

We have a table with high total CPU use that is accessed frequently (millions of executions), though each execution only takes about 8 ms. Depending on how many rows are returned, this could be normal.

We explain the statement, using the explain (wrench) button in Figure 127.

![Figure 128: YGAS explain](image)

In Figure 128, note that there are two local predicates (MANDT= and FLAG=), but only one column in the SQL is matched in the YGAS~0 index, since “#key columns” is 1. We drill in on the table name to check the table and its indexes as shown in Figure 129.
There is only one index, where FLAG is the last column in the index. Using the index, DB2 can match the first index column MANDT, and screen FLAG, but the entire index must be read, since there is only one value of MANDT. The index is small (only about 15K entries), but since FLAG has 182 distinct values, it would be more efficient to be able to do matching index access for both the MANDT and FLAG columns.

We saw above in Figure 70 how one can display the optimizer details to confirm that the local predicates are being screened on the index.

In this case, as outlined in our decision matrix in Section 8.1.6 for a custom program and custom table we create a new index (MANDT, FLAG) on YGAS.

8.1.12. Performance problem – Slow SQL due to LOB columns

With DB2 9.7, LOBs can be stored in-line in table rows. This allows them to be buffered in DB2 buffer pools, as is described in SAP note 1351160. For situations where the LOB cannot be stored in-line (e.g. when using DB2 9.5, or when the LOB columns are too long to be stored inline), I/O to retrieve the LOB column can impact SQL performance.

© Copyright IBM Corporation, 2012
In Figure 130, note that the statement takes 23 ms and it reads on average one table row for each execution (80,954 rows read in 80,951 executions). This is slow performance to retrieve one row. We can tell from these statistics that the slow SQL is not caused by applying local predicates on the table, since that would cause high rows read. We can explain the statement to see the access path.

Figure 131 shows there are only two local predicates, and they are both applied in the index (key columns is two). We now know that inefficient index access is not the problem. Notice also that the SELECT is retrieving all the columns from the table by using SELECT *.
Why is the statement so slow? We display the table columns with SE11. Note that CRYPTIC_NUMBER is SAP data type RAWSTRING.

Figure 132: CCSEC_ENC SE11

In Figure 132, select Utilities > Database Object > Display to display the database definition of the table shown in Figure 133.

Table CCSEC_ENC: Display Database Object

We see that DB2 defines the CRYPTIC_NUMBER column as data type BLOB.

In order to reduce the I/O activity for reading the LOB object, the table can be moved to its own tablespace where the tablespace is configured to be buffered in AIX filesystem cache. DB6CONV can be used to move the table online. This is described in SAP note 1353421.

Another alternative may be to change the SQL so that it does not retrieve the CRYPTIC_NUMBER column. The programmers can verify if the statement can be modified to list the columns that are needed, rather than using “SELECT *” for all the table columns.
8.2. **Create candidate list with ST04 Tables**

The ST04 SQL cache display does not have a reset/since reset function to view SQL for only a recent time period. ST04 Tables does have “reset” and “since reset”, so it can be used in conjunction with ST04 SQL cache. ST04 Tables can be used to find the tables with the most rows read. Since high numbers of rows read is a symptom of SQL where the local predicates are evaluated on the table rows (not in an index), one can then filter the SQL cache to search for only statements referencing the tables with high rows.

![Figure 134: ST04 Tables](image)

![Figure 135: ST04 Selection Criteria](image)

For instance, when displaying the SQL Cache, one can enter a text string, e.g. *BSIP*, to see all the statements with BSIP in the SQL text.

An alternative, when displaying the full SQL cache, one can enter the filter in the following way.

![Figure 136: ST04 SQL cache set filter](image)

© Copyright IBM Corporation, 2012
Select “Set Filter”, as shown in Figure 136.

![Figure 137: Cache Filters](image1)

Select “SQL Text”, and move it to Filter Criteria on the left. Then, press “Determine Values for Filter Criteria”.

![Figure 138: Cache Filter Values](image2)
Enter the values, and press execute.

8.2.1. ST04 Tables and TEMP table activity

On a transactional system such as ECC, it is unusual to have TEMP tables as the most frequently accessed tables in the system. This can be a symptom of a problem. When ST04 Tables over an interval shows TEMP tables as the most frequently accessed tables, here is how to find the SQL that is causing the TEMP table reads.

![Figure 139: ST04 Table Snapshot shows TEMP tables as top](image3)

© Copyright IBM Corporation, 2012
If the SQL is still running, it can be found using ST04 Applications snapshot – find the “Appl Handle” in the ST04 Applications snapshot in Figure 140 that matches the number in the “Table Schema” in Figure 139.

![Figure 140: ST04 Applications snapshot](image1)

Press Display (magnifying glass) in Figure 140, and then select the “Statement Text” tab.

![Figure 141: ST04 Applications Statement Text](image2)

Now, we can either display the source ABAP, or explain the statement to see the access plan to determine why the TEMP table accesses are so high.
We display the ABAP in Figure 142, and it is SAP code.

In Figure 143, note that DB2 has chosen an access path using TEMP table. We know that the SQL is doing a large number of reads on TEMP from ST04 Tales. If TEMP is being read frequently, we can infer that the problem may be that the number of rows selected from FAGLFLLEXA is larger than DB2 expected. This may be a problem caused by prepare with markers, since prepare with markers can cause DB2 estimates to be wrong when there is data.
skew, or when using range predicates in the SQL. We will need to trace the SQL to determine the values used at runtime.

When we get the values, and explain the statement using values, we see that DB2 now chooses a different access path.

![Figure 144: High TEMP reads SQL explain with values](image)

The SQL has local predicates on both tables, so HSJOIN Figure 144 looks like it might be a better choice than the original access path. Since this is SAP ABAP code, we can search SMP or open a message to SAP.

You can also fine excessive TEMP table activity in the SQL cache.

![Figure 146: SQL cache TEMP activity](image)

To add optional columns to the ST04 SQL cache display, select “Change Layout”.

© Copyright IBM Corporation, 2012

Page 91
In Figure 147, select the columns for Temp activity, and press the left arrow to move them to the “Displayed Columns” area. Then press execute.

We can sort the cache statements in Figure 148 by “Temp data logical reads”, to find the high impact statements using TEMP tables.

And we find the same statement that was active in the ST04 Applications snapshot.
8.3. **Create candidate list from ST03N**

To find transactions and programs that may have inefficient DB access, run ST03N, sort by total database time.

![Load Analysis in System](image)

**Figure 150: ST03N by DB time**

Transactions which have the longest DB time, and thus probably the largest impact on the database server, are at the top of the list. Next, click on the “Database” tab, to see the per-call times.
Figure 151: ST03N by DB time database tab

Check the average times for sequential read (‘0 Seq read time’ column). If this is long, it usually means one of two things:

- Each read retrieves many rows (this is not a problem, if the program needs the data), or
- The SQL access is inefficient, and it takes a long time for a few rows (this is a problem).

At this point, we cannot tell which is occurring, but one can trace the programs with ST05, as shown in Section 7.4, to determine whether there is a problem or not.
8.4. **ST03N with table statistics**

If the SAP parameter stat/tabrec is enabled, then SAP will gather table call statistics for each dialog step. These statistics can be displayed, to determine which table accounts for the long DB time for a transaction or program.

![Table Access Statistics](image.png)

Figure 152: ST03N table call statistics

In this example, RFEBBU00 calls to AVIP are slow – on average it retrieves 9 rows per second. Either the program makes many calls to AVIP that return no rows, or the SQL is slow. As described in Section 8.2, we can use ST05 to trace the program. By limiting the trace to AVIP, it is less likely that the trace will wrap causing the slow calls to be lost.

9. System Health Check

9.1. **CPU activity**

As discussed above, SQL problems can contribute to high CPU use on the DB server, and ABAP coding problems can contribute to high CPU use on the application server. If the system has high CPU utilization, then first search for SQL and application problems that may be causing high CPU use.

For native or dedicated LPAR environments, SAP ST06 (or OS07) transaction has statistics on recent use, and on historical CPU use. Since ST06 has statistics based on hourly averages, it will not show CPU constraints until they are very severe. It is generally better to monitor CPU use using an AIX based tool, and gather statistics on 5 or 10 minute intervals, in order to be able to calculate average and peak activity.

For SPLPAR environments, ST06N (or OS07N) is required together with an updated saposcol, in order for SAP to correctly report CPU activity. See SAP note 994025 regarding SAP and AIX requirements for SAP monitoring in a virtualized AIX environment.
### Important Note: The older OS monitor transactions and old versions of saposcol will give misleading and incorrect output, when used in a virtualized SPLPAR AIX environment.

<table>
<thead>
<tr>
<th>Server</th>
<th>_LRP1_02</th>
<th>Fri Oct 30 15:59:59 2009</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>AIX</td>
<td>3.6.0.0C117E34</td>
</tr>
<tr>
<td>CPU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Value Unit</td>
<td>Description</td>
</tr>
<tr>
<td>User utilization</td>
<td>96 %</td>
<td>Number of CPUs</td>
</tr>
<tr>
<td>System utilization</td>
<td>3 %</td>
<td>Average processes waiting (1 min)</td>
</tr>
<tr>
<td>Idle</td>
<td>1 %</td>
<td>Average processes waiting (5 min)</td>
</tr>
<tr>
<td>IO wait</td>
<td>0 %</td>
<td>Average processes waiting (15 min)</td>
</tr>
<tr>
<td>System calls</td>
<td>1,088 /s</td>
<td>Context switches</td>
</tr>
<tr>
<td>Interrupts</td>
<td>339 /s</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host system</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Value Unit</td>
<td>Description</td>
</tr>
<tr>
<td>Model</td>
<td>IBM,0116-FHA</td>
<td>Processor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Frequency</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Virtual system</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Value Unit</td>
<td>Description</td>
</tr>
<tr>
<td>Partition type</td>
<td>Shared Pool LPAR</td>
<td>Pool utilization authority</td>
</tr>
<tr>
<td>SMT mode</td>
<td>Off</td>
<td>Shared Pool CPUs</td>
</tr>
<tr>
<td>Threads</td>
<td>2</td>
<td>Physical CPUs consumed</td>
</tr>
<tr>
<td>Capped</td>
<td>Off</td>
<td>Physical CPUs idle</td>
</tr>
<tr>
<td>Virtual CPUs</td>
<td>8</td>
<td>Available capacity</td>
</tr>
<tr>
<td>Entitlement</td>
<td>1.00 CPUs</td>
<td>Available capacity consumed</td>
</tr>
<tr>
<td>Entitlement consumed</td>
<td>132,1 %</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Value Unit</td>
<td>Description</td>
</tr>
<tr>
<td>Physical memory</td>
<td>38,706,160 KB</td>
<td>Free physical memory</td>
</tr>
<tr>
<td>Pages in</td>
<td>0 /s</td>
<td>Pagged in</td>
</tr>
<tr>
<td>Pages out</td>
<td>0 /s</td>
<td>Pagged out</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Swap size</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Value Unit</td>
<td>Description</td>
</tr>
<tr>
<td>Configured swap size</td>
<td>33,554,432 KB</td>
<td>Maximum swap size</td>
</tr>
<tr>
<td>Free swap size</td>
<td>32,881,780 KB</td>
<td>Actual swap size</td>
</tr>
</tbody>
</table>

**Figure 153: ST06N with AIX virtualization support**

In Figure 153, note that the CPU utilization is reported as nearly 100%, but this LPAR is not capped and there is actually quite a bit of CPU capacity that is still available to the LPAR — there are over 36 physical CPUs idle. The LPAR is defined with 6 Virtual CPUs, and so if there is sufficient available CPU capacity in the shared pool, CPU on this LPAR could go up as far as 600% of entitlement, since the entitlement is only one CPU. Thus, the LPAR in Figure 153 is running over its entitled capacity, but it is not CPU constrained at this time.
When the shared pool is exhausted, then there is a true CPU constraint. In Figure 154, there is a CPU constraint – the shared pool CPUs are completely utilized with only 0.02 idle. Each LPAR on this system will get its entitled CPU, but there is not more CPU available to support additional demand.

<table>
<thead>
<tr>
<th>Partition type</th>
<th>Shared Pool LPAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pool utilization authority</td>
<td>Granted</td>
</tr>
<tr>
<td>SMT mode</td>
<td>On</td>
</tr>
<tr>
<td>Shared Pool CPUs</td>
<td>8.00</td>
</tr>
<tr>
<td>Threads</td>
<td>2</td>
</tr>
<tr>
<td>Physical CPUs consumed</td>
<td>2.92</td>
</tr>
<tr>
<td>Capped</td>
<td>Off</td>
</tr>
<tr>
<td>Physical CPUs idle</td>
<td>0.02</td>
</tr>
<tr>
<td>Available capacity</td>
<td>2.94 CPUs</td>
</tr>
<tr>
<td>Available capacity consumed</td>
<td>98.8 %</td>
</tr>
<tr>
<td>Virtual CPUs</td>
<td>4</td>
</tr>
<tr>
<td>Entitlement</td>
<td>1.40 CPUs</td>
</tr>
<tr>
<td>Entitlement consumed</td>
<td>137.3 %</td>
</tr>
</tbody>
</table>

**Figure 154: Shared Pool Consumed**

In order to monitor the shared pool utilization, the LPAR must be defined with ‘Pool Utilization Authority” (PUA). As shown in Figure 155, if PUA is not enabled, then SAP ST06N cannot report CEC Pool utilization. In Figure 155, the LPAR is using nearly 100% of its entitled capacity, so if other LPARS were at their entitlement, and if the shared pool were at 100% utilization, then this LPAR would not be able to get any more CPU resources. But, we cannot tell without Pool Utilization Authority enabled for the LPAR in the HMC.

**Figure 155: ST06N PUA not enabled on LPAR in HMC**
An LPAR can be CPU constrained by the number of VCPUs, even when there is CPU capacity available in the shared pool. Note that in Figure 156, the LPAR is consuming about 2x its entitlement. The entitlement is 1 CPU, and there are two Virtual CPUs, so the LPAR can never use more than 200% of entitlement. If there is a lot of variation in CPU use on the application server, it is better to define more VCPUs, and let AIX manage the shared pool CPU allocation based on LPAR weights, so that if CPU is available then an LPAR can temporarily use more than its entitlement.

One can use nmon, sar, ptx, or other tools to gather and report on historical utilization.

While it may be acceptable for an application server running batch work to run at 100% utilization, if an application server supporting dialog or critical interfaces is frequently hitting 100% utilization, it can have a significant impact on response time and end-user satisfaction.

If the DB server has high CPU activity:

- Evaluate the impact of inefficient SQL and applications as in Section 8.1.
- If there is a SAP application instance on the DB server, investigate whether it can be moved off.
- If no work can be moved off, evaluate moving some batch jobs to run outside of peak time.
- Acquire more CPU power for the DB server

If one application server has high CPU activity

- Evaluate the impact of inefficient ABAP code on excessive CPU use as in Section 7.2.
- Evaluate changes in the SAP login groups to balance the workload differently
- Change the mix of batch/update/dialog to reduce the amount of work on the server

© Copyright IBM Corporation, 2012
Acquire more CPU power

9.2. I/O activity

One can get an overview of I/O activity from ST04. If symptoms of problems are seen, then more detailed analysis (such as using iostat or sar in AIX, or monitoring programs in the disk system) would be needed.

9.2.1. Good I/O performance in DB2

In Figure 157, the average I/O response times for each tablespace are very good – below 5 ms. Also, each tablespace has roughly similar response times.

![Figure 157: ST04 tablespace good I/O times](image)

9.2.2. Symptom of I/O hotspot on disk

Figure 158 (which is ST04 downloaded to excel) shows the symptoms of having a “hot spot” with slow performance on some of the disks. In this case, one of the tablespaces has average read times of nearly 10 ms, while the others have response times of 3-4 ms. This symptom is usually caused by having a DB layout where an active tablespace is not spread across enough disks. The cure is usually to review the layout of the DB, and spread the tablespaces across many disks, either by making changes at the DB level (spread tablespace over more containers), AIX level (spread filesystem across more disks), or in the disk system (spread active LUNs apart on physical disks).

![Figure 158: ST04 (downloaded to excel) variation in response time](image)

9.2.3. Symptom of slow I/O

In Figure 159 average read times are slow on all tablespaces. This is a symptom of a system-wide constraint. One would need to check AIX, DB2 and SAP settings related to I/O, check disk performance statistics, and if necessary open an AIX performance PMR.
9.3. **DB2 Review**

The document “SAP DBA cockpit”, which describes many SAP DB2 monitoring capabilities that are not discussed in this paper, can be downloaded from [http://bit.ly/DB2_SAP_PDF](http://bit.ly/DB2_SAP_PDF).

### 9.3.1. Database Hit Rate

ST04 can be used to review the hit rate in the database. Since SAP often executes many SQL operations per dialog step, the goal should generally be to have a hit rate in the high 90s. A 96% hit rate is two times the misses of a 98% hit rate. A 94% hit rate is three times the misses of a 98% hit rate. The impact of cache misses on response time quickly grows as hit rate decreases.

Since inefficient SQL can cause the symptom of low hit rates or of high hit rates, the SQL cache analysis process in Section 8.1 should be the first action, before evaluating the buffer size and increasing buffer memory. After the SQL analysis, one can review adding more memory to DB2 (in order to increase hit rate) by using the process in Section 9.6.4.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2 PRD#BTABD_L</td>
<td>85.15</td>
<td>2,757,452,845</td>
<td>409,445,231</td>
<td>8.76</td>
<td>9.27</td>
</tr>
<tr>
<td>3 PRD#STABD</td>
<td>96.88</td>
<td>10,013,752,063</td>
<td>312,331,312</td>
<td>8.76</td>
<td>8.63</td>
</tr>
<tr>
<td>4 PRD#BTABD_4</td>
<td>74.84</td>
<td>765,690,778</td>
<td>192,660,114</td>
<td>9.40</td>
<td>9.50</td>
</tr>
<tr>
<td>5 PRD#BTABD_3</td>
<td>97.79</td>
<td>7,497,860,831</td>
<td>165,635,491</td>
<td>9.33</td>
<td>9.33</td>
</tr>
<tr>
<td>6 PRD#BTABD_3</td>
<td>85.92</td>
<td>910,147,771</td>
<td>128,160,492</td>
<td>7.39</td>
<td>7.60</td>
</tr>
<tr>
<td>7 PRD#MSEGDD</td>
<td>96.67</td>
<td>2,113,509,096</td>
<td>70,320,867</td>
<td>9.65</td>
<td>10.04</td>
</tr>
<tr>
<td>8 PRD#BTABD_1</td>
<td>95.41</td>
<td>1,211,763,918</td>
<td>55,612,890</td>
<td>7.76</td>
<td>9.98</td>
</tr>
<tr>
<td>9 PRD#BTABD_5</td>
<td>83.41</td>
<td>223,309,650</td>
<td>37,046,175</td>
<td>9.23</td>
<td>9.75</td>
</tr>
<tr>
<td>10 PRD#MSEGDD</td>
<td>91.51</td>
<td>398,265,123</td>
<td>33,828,202</td>
<td>9.17</td>
<td>9.17</td>
</tr>
</tbody>
</table>
If there are many SQL and ABAP problems in the system, hit rate in DB2 is not meaningful. Figure 160, for example, is taken from a system that had a number of inefficient SQL statements, where DB2 did scans of tables or indexes, since the SQL could not be processed efficiently in indexes. The tables were moderately large, but frequently accessed, so they stayed in memory. This caused a very large increase in data reads relative to index reads, and created a hit rate that did not give meaningful information about I/O activity for the average transaction, since the hit rate was artificially increased by the few transactions that repeatedly scanned the in-memory tables.
9.4. SAP buffer settings

When SAP buffers fill and swap, it can cause performance problems for transactions and batch jobs. One example of this was shown in Section 8.1.9, where the generic buffer area was filled. In addition to this, another common problem is when the program buffer fills, causing swaps and impacting response time.

The impact of SAP buffers being too small is clearest in the DB accesses column, where one can see how many calls are made. In this example, the DB call impact of most of the buffer types is small, while there are millions of calls to the DB for tables that should have been in the Generic Key buffer.

![Figure 161: ST02 swaps](image)

© Copyright IBM Corporation, 2012
If ST02 shows program buffer swaps and accesses, check ST03N, to evaluate response time impact.

Figure 162: ST03N overview

Figure 162 shows ST03N data that has been downloaded to Excel. Note that load and generate time is about 5% of average dialog response time. However, this impact will not be evenly distributed. When the programs and screens referenced by a program are in SAP buffers, the program will run quickly. If they are not present, then it might take several seconds longer than normal.

If the application servers are memory constrained, then one may choose to live with this impact. If one wants to be able to have more reliable transaction response times, then one would increase the size of the buffers in SAP.

9.5. SAP buffered table statistics

There are two quick checks that can be done:
- Tables that are not buffered, but which should be, and
- Tables which are buffered, but whose settings should be changed.

9.5.1. Not buffered but could be buffered

The most common problem with buffered tables is tables that could be buffered, but are not. Generally, these are custom tables. Most SAP tables have the appropriate settings already enabled.
Figure 163 is a segment of an ST10 report for not buffered tables, ordered by calls to the database.
Impact on the database is more a function of total pages referenced and not DB calls, but calls or rows are the only orders available with ST10, since ST10 cannot determine whether SQL is executed efficiently or inefficiently in DB2.

Figure 163: ST10 not buffered tables

There are three tables in this list that may be buffering candidates:
- YMSESPIV02
- OICDC
- YVGMDRCADDR

Since OICDC is a SAP standard table (it does not start with Y or Z), one can check in SE13 if buffering is allowed but not currently enabled. If it is allowed but not enabled, we can turn buffering on.

The two Y* tables are custom tables, so we must check:
- How large is the table
- Will it be changed frequently (it looks like not, from ST10 stats)
- Can the application tolerate a short period where the buffered versions are different, when the table is changed in the database and changes are propagated to application servers.

Since the tables are all read with direct read, if the table is reasonably small (e.g. <5 MB), not changed much, and the application can tolerate inconsistency, then they could be fully buffered. If the tables are very large, then they might be single record buffered, to save buffer space by buffering only the referenced rows.

© Copyright IBM Corporation, 2012
ABAP SELECT SINGLE (SAP direct read) can read from either the generic or single record buffer. ABAP SELECT can only read from the generic buffer.

9.6. **Evaluate memory in AIX**

There are three different SAP instance memory management options, which are described in more detail below. The latest version supported for your SAP kernel should be used. The three options are

- **ES/TABLE = UNIX_STD** – should not be used with AIX 5 or AIX 6
- **ES/TABLE = SHM_SEGS** – for SAP kernels w/o support for `ES/SHM_SEGS_VERSION=2`
- **ES/TABLE = SHM_SEGS & ES/SHM_SEGS_VERSION=2** – latest and best version

See below for information on kernel levels required for each.

SAP Instance memory parameter settings can have an impact on AIX memory use. For example, the latest version of SAP EM (using `ES/SHM_SEGS_VERSION = 2`) has more efficient disclaim processing, which can reduce the amount of real memory needed for SAP EM. If there is paging in AIX, please review the latest versions of the SAP notes referenced in Section 9.6.2.

9.6.1. **AIX 6.1 Restricted Tunables concept**

Since AIX 5.3, six tuning commands (vmo, ioo, schedo, raso, no and nfso) are available to tune the operating system for the specific workload. Beginning with AIX 6.1, some tunables are now classified as restricted use tunables. The default values for restricted tunables are considered to be optimum for most AIX environments and should only be modified at the recommendation of IBM support professionals. As these parameters are not recommended for user modification, they are no longer displayed by default, but can be displayed with the –F option (force). The –F option forces the display of restricted tunable parameters when the options –a, -L, or –x are specified alone on the command line to list all tunables. When –F is not specified, restricted tunables are not included in a display unless specifically named in association with a display option.

9.6.2. **AIX Alternative Memory Management**

In the previous versions of SAP kernels (4.XX and 6.XX), extended memory bottleneck would occur if the high numbers of user contexts existed on the system. To alleviate that problem, SAP implemented a new AIX memory management (`ES/TABLE=SHM_SEGS`) that allowed individual user context and overall extended memory size to grow beyond several GB. New memory management was based on the several different AIX memory allocation functions like disclaim(), late swap allocation etc… Several benefits are achieved using this method:

- Eliminating potential memory bottlenecks,
- User context change is much faster, particularly for large contexts,
- A much higher throughput is achieved, especially for higher context change rates (local updates …)
Variable PSALLOC=EARLY must never be set if this Extended Memory implementation is activated. The profile parameter that would initiate the utilization of the new Extended Memory Management is:

- **ES/TABLE=SHM_SEGS**

Additional parameters are required and important for the successful implementation of the new memory management:

- **EM/TOTAL_SIZE_MB**: allows a maximum of 16 GB extended memory and can be larger, if required. This can be used to limit the total amount of Extended Memory, to prevent a swap bottleneck.
- **ES/SHM_PROC_SEG_COUNT**: specifies the number of shared memory segments that are used as shared and privately.
- **ES/SHM_MAX_PRIV_SEG**: specifies the number of shared memory segments that are used privately.
- **ES/SHM_USER_COUNT**: the maximum number of users in all open sessions.
- **ES/SHM_SEG_COUNT**: the maximum number of AIX system segments that ESSHM uses.
- **ES/SHM_SEG_SIZE**: size of the AIX system segments (in megabytes)

To properly implement the AIX alternative memory management following notes should be reviewed:

- OSS Note: 445533 (up to SAP 4.6D),
- OSS Note: 789477 (as of 6.10).

In the specific situation that is primarily related to the large memory requirements based on the new AIX memory management, the memory management performance anomaly can occur. This is primarily related to the method that is used to map and disclaim the AIX memory pages used by the SAP kernel. To improve the performance of the memory management the following variable should be included in the profile of the associated application server:

- **ES/TABLE = SHM_SEGS**
- **ES/SHM_SEGS_VERSION = 2**

To get more details about the memory mgmt. process used in this case, please check the following notes:

- OSS Note: 856848,
- OSS Note: 1088458.

### 9.6.3. AIX paging

The standard tools (nmon, ptx, sar) can be used for monitoring paging. The page space page-ins and page-outs (paging to disk) are the important indicator to check.
If you encounter AIX paging, a quick check would be to look at the virtual memory manager parameters. AIX memory pages are categorized as being computational or file pages. AIX will always try to utilize available physical memory (subject to vmo parameter settings). What is not required to support computational page demand will tend to be used for filesystem cache. In AIX 5L, a number of the default “vmo” settings in AIX 5.3 are not optimized for DB2 database workloads and should be modified. (The defaults have been changed in AIX 6.1 and should not require modification.)

Check the settings for the following vmo parameters using the vmo –a command (as root user):
- minperm% = 3
- maxperm% = 90
- maxclient% = 90
- lru_file_repage = 0

Refer to SAP Note 973227 - AIX Virtual Memory Management: Tuning Recommendations.

Note: In AIX 5.2 environments with large physical memory, set maxperm%, maxclient% such as not to exceed 24GB of memory.

If you encounter AIX paging on a system with an SAP instance, and you have reviewed that the correct EM configuration is used, consider paging first as a symptom of an application problem, and approach it from the application statistics in SAP. Since SAP memory use can vary greatly, as programs allocate and free SAP memory, ABAP problems such as the slow internal table access described in Section 7.2 can contribute to paging problems. If a report processing many line items runs quickly, while it is running it will acquire more memory, and then release it. If the program runs too long because of inefficient ABAP coding, then the program will need the memory for longer than it should. When several programs do this, it can cause paging. If the programs were coded efficiently, then they would quickly finish, and this would reduce the likelihood that they would all be running simultaneously.

After having found the programs and users with large memory requirements, one can consider running the reports in batch on a server specially configured for large memory requirements, or the huge reports can be automated to run at night, when there may be less demand for memory.

If the total SAP demand for memory is larger than available memory, one can reduce SAP memory quotas for EM and heap, but if the workload is unchanged, this will cause programs to go into PRIV mode (when EM is exhausted) or to abend when running out of heap.
Check for running programs using large amounts of memory with ‘SM04 > goto > memory’:

![Image of Block List](image1)

**Figure 164: SM04 > goto > memory**

If a user is running several transactions, only one of the transactions will be displayed in SM04. Choose a user, and press “Sessions” to see the names of the transactions.

![Image of Overview of Sessions](image2)

**Figure 165: SM04 Sessions**

One can then examine these transactions further to determine which use the memory. Use table TSTC to see transaction name and code. VA05 is “List of Sales Orders”.

© Copyright IBM Corporation, 2012
ST03N has historical statistics on memory usage.

Figure 166: ST03N memory use statistics

9.6.4. Evaluating increasing memory for DB2 or SAP

If you have evaluated SAP or DB2 memory use, and would like to increase the buffer sizes in DB2 or SAP, how do you determine whether there is memory available, or whether adding memory will cause paging? The free page information on commands such as vmstat does not always reflect how much memory is really available to be added to SAP or DB2 buffers.

If vmo shows free memory, then there are truly free memory pages that can be added to SAP or DB2 buffers. Check vmstat (or ST06) over the course of several days, and if it consistently shows free memory, then use the minimum free pages to determine the amount of memory that is available to add to DB2 or SAP. Do not try to allocate all free memory to SAP or DB2.
There are situations, such as a database server not using DIO or an NFS server where files are mapped to memory, where ST06 or AIX vmstat may show almost no free pages, but there is still a lot of memory that can be added to SAP or DB2.

Unless a file is opened for DIO (direct I/O) AIX maps file pages into available physical memory, in order to help performance. There are AIX parameters on the vmo (AIX 5 vmo) command that set the limits of memory-mapped files. On a database server, first check that SAP note 78498 or 973227 (depending on SAP and AIX version) has been implemented to establish limits on real memory used for memory-mapped files.

Check the amount of real memory that is being used for memory-mapped files. This can be done with the command svmon –G.

![Operating System Monitor: AIX](image)

Figure 167: ST06 displays free memory

<table>
<thead>
<tr>
<th>Local</th>
<th>Operating System Monitor: AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td></td>
</tr>
<tr>
<td>Utilization</td>
<td>user %</td>
</tr>
<tr>
<td></td>
<td>system %</td>
</tr>
<tr>
<td></td>
<td>idle %</td>
</tr>
<tr>
<td></td>
<td>io wait %</td>
</tr>
<tr>
<td>System calls/s</td>
<td>2,963</td>
</tr>
<tr>
<td>Interrupts/s</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical mem avail Kb</td>
<td>10,405,616</td>
</tr>
<tr>
<td>Physical mem free Kb</td>
<td>3,300,492</td>
</tr>
<tr>
<td>Pages in/s</td>
<td>0</td>
</tr>
<tr>
<td>Pages out/s</td>
<td>0</td>
</tr>
</tbody>
</table>

Kb paged in/s | 0
Kb paged out/s | 0
The total of AIX memory mapped pages is the sum of “pers” and “clnt” columns, for both “pin” and “inuse” rows. In this example, 234+1,585,658=1,585,892

If you are not using JFS2 and DIO to read DB2 tablespaces, rather than having DB2 have a cache miss and do I/O that AIX fulfills from memory-mapped file cache, it is usually more efficient to give the memory to DB2 buffers. Likewise, if additional SAP buffer memory can reduce database calls, then it is generally good to give the memory to SAP.

Since SAP and DB2 memory use can vary widely from hour to hour, run the svmon command periodically for several days or a week, to determine the minimum number of (persistent + client) pages over the period. This minimum (pers+clnt pin+inuse) should be compared with vmo (vmo) maxperm, to determine if there is available memory.

Make changes gradually, and don’t over-allocate the memory in DB2 and SAP buffers, as that can cause AIX paging.

Don’t add memory to DB2 or SAP just because AIX shows that there are available pages. If DB2 hit rates are low, or SAP buffers need to be increased to support the workload, then increasing the size of the DB2 and SAP buffers is reasonable. If application performance is good, and DB2 hit rates are good, then the extra memory could be given to another LPAR.

10. Four guidelines for avoiding performance problems

As seen in the examples above, there are a few general rules for avoiding performance problems.

10.1. Use the SAP data model

If you’re evaluating the performance of custom code, and it runs slowly because the predicates don’t match the indexes on SAP tables, the odds are very good that the program is looking for the data in the wrong place. Whenever you see inefficient SQL with a local predicate on a document number, but the document number is not indexed on the table, it is almost always a data model usage problem. Most SAP business documents (e.g. sales order, purchase order, delivery note, etc) can be found using the document number with a standard SAP table and standard SAP index.

In addition to the examples in Section 8.1.8 and Section 8.1.9, See SAP notes 185530, 187906, and 191492 for examples of incorrect and correct use of the SAP data model.

The symptom of this problem is in the ST04 SQL cache (high elapsed time or rows per execution) or in ST05 (low rows selected per second) This happens when the local predicates in the SQL do not
match the index columns on the table and DB2 has to read many blocks of table data to retrieve the result. One has to examine the SQL to confirm the cause.

**10.2. Use array operations on the database**

If the program builds internal tables that contain keys for selects on other tables, evaluate whether an array operation such as FOR ALL ENTRIES can be used to perform array selects, rather than using LOOP AT with individual database calls.

The symptom of this problem is seen in ST05 traces, where a program makes frequent calls to a table, and each call accesses few rows.

**10.3. Check whether the database call can be avoided**

There are several versions of this problem:

- A table is set as buffered, but the application server generic or single record buffers are too small to hold the table rows. The cure for this is to increase the size of the SAP buffers. See Section 8.1.10.
- A table is not set to be buffered, but should be. In this case, the table is usually read-only, and the application can tolerate a small interval when the data is not in synch on all the application servers. In this case, the table attributes should be changed to buffered. See Section 7.4.5.
- A program is repeatedly fetching the same information from the database. This problem can be detected by using the “identical selects” report in ST05. The program needs to be examined to see how it can be changed so that it does not have to repeatedly go back to the database for the same information. See Section 7.4.1.6.5.

**10.4. Write ABAP programs that are line-item scalable**

If the program will process many lines in a report:

- Use BINARY SEARCH on the “READ TABLE from itab” statements. See Section 7.3.1.
- Evaluate whether internal tables need to be defined as SORTED. See Section 7.3.2.

The symptom of this problem is high CPU use for a program, where CPU use does not scale with additional line items – e.g. a 100 line report takes 1 second CPU, but a 1000 line item report takes more than 10 seconds CPU, and a 10,000 line report takes much more than 100 seconds CPU. These scalability problems get worse as the report lines increase.
11. Appendix 1: summary of performance monitoring tools

A quick summary of key tools and their main functions in performance monitoring follows.

11.1. SAP

11.1.1. DBACOCKPIT

New transaction which can be used for all DBA related activities. It contains functions that are separately available via ST04, DB02, etc. As mentioned above, detailed information on DBA Cockpit is available at http://bit.ly/DB2_SAP_PDF.

11.1.2. DB02 Transaction

DB02 is used to display information about tables and indexes in the database, such as space usage trends, size of individual tables, etc.:

- Display all indexes defined on tables (DB02 > single table analysis > enter table name > index tab)
- Check column cardinality, i.e. the number of distinct values in a column (DB02 > single table analysis > enter table name > table tab)

11.1.3. SE11 Transaction

SE11 is used to gather information about data dictionary and database definition of tables, indexes, and views:

- Display table columns and indexes (SE11 > enter table name > display > extras > database objects > check)
- Display secondary indexes defined in data dictionary (SE11 > enter table name > display > indexes). There may be data dictionary indexes that are not active on the database.
- Display view definitions

11.1.4. SE30 Transaction

When STAD or ST03N shows that most of a program’s elapsed time is CPU, SE30 is used to investigate where CPU time is spent in an ABAP program

11.1.5. SM12 Transaction

SM12 > extras > statistics can be used to view lock statistics:

- High percentages of rejects can point to a concurrency problem (multiple programs trying to enqueue the same SAP object) that may be solved via SAP settings such as “late exclusive material block” in the OMJI transaction. There are different SAP settings for different parts of the business processes, so these changes would be implemented with SAP functional experts.
- High counts of error can point to a problem where the enqueue table is too small. Compare “peak util” with “granule arguments” and “granule entries” to check for the table filling.
11.1.6. SM50 Transaction
SM50 is an overview of activity on an SAP instance. If many processes are doing the same thing (e.g. access same table, ENQ, CPIC, etc), this can point to where further investigation is required.

11.1.7. SM51 Transaction
SM51 can be used to check instance queues (goto > queue information)
- If there are queues for DIA, UPD, UP2, etc, there will be “wait for work process” in STAT and ST03. Queues can be caused by performance problems, or by a shortage of work processes. Check performance issues first. If the root cause is a DB or ABAP performance problem, adding work processes can make performance worse by overloading the system.
- If there are queues for ENQ, there is a problem with enqueue performance.

11.1.8. SM66 Transaction
SM66 gives an overview of running programs on an SAP system. If many processes are doing the same thing (e.g. access same table, ENQ, CPIC, etc), this can point to where further investigation is required.

11.1.9. STAD Transaction
Is used to displays STAT records for an interval from all instances on an SAP system.

11.1.10. ST02 Transaction
ST02 is used to monitor the activity in SAP managed buffer areas, such as program buffer, generic buffer, roll, and EM.

11.1.11. ST03N Transaction
ST03N is not a tool for solving performance problems. It is a tool that is mainly useful for tracking historical activity. One can monitor average response times for individual transactions and for the system as a whole, and get counts of dialog steps to use for trend analysis.

There are a few limited ways that it might be used in performance monitoring:
- As a filter for inefficient programs. Use the ST03N “transaction” profile, sort the list by elapsed time, and look for transactions which use very little CPU relative to elapsed time, e.g. 10% or less of elapsed time is CPU on the application server. These may have problems such as inefficient database access, slow RFC calls, etc.
- As a filter for problems that occur at a certain time of the day. Run ST03, and select “dialog” process display. Use the ST03N “times” profile, press the right arrow to go to the screen that displays average direct read, sequential read, and change times. Look for hours of the day when the average time goes up. This could point to a time when there is an I/O constraint, or CPU constraint on the DB server.
- As a filter for ABAP problems. Sort the “transaction” profile by elapsed time, and look for transactions where CPU is a large majority of elapsed time. These may have problems with handling internal tables inefficiently.
11.1.12. **ST04 Transaction**
ST04 has many functions, the most important for performance are: viewing the SQL cache, and monitoring database buffer activity and monitoring DB2 application processes.

11.1.13. **ST05 Transaction**
ST05 is one of the most important tools for SAP performance, among its functions are:
- Trace calls to database server to check for inefficient SQL when program is known.
- Compress and save SQL traces for regression testing and comparisons.
- Trace RFC, enqueue, and locally buffered table calls.

11.1.14. **ST06N Transaction**
Display AIX-level stats for the application server – paging, CPU usage, and disk activity.

11.1.15. **ST10 Transaction**
ST10 is used to monitor table activity, and table buffering in SAP on the application server:
- Check for tables that are candidates for buffering in SAP
- Check for incorrectly buffered tables

11.1.16. **RSINCL00 Program**
Expand ABAP source and include files, with cross-reference of table accesses. This is useful when examining ABAP source, as it gives an overview of the whole program.

11.1.17. **SQLR Transaction**
Merge an ST05 trace with STAT records, to determine which dialog step executed which statements. This is useful when tracing a transaction made up of many dialog steps, to join the trace to the dialog step that issued the problematic SQL.

Depending on the SAP version, this may be available as a transaction SQLR, program /SQLR/0001, or program SQLR0001.

11.1.18. **RSTRC000 Program**
Lock a user mode into a work process. This is useful for doing traces (such as OS level or DB2 level traces) where one needs to know the PID to establish the trace. Once the user is locked into a work process, one can determine the PID or the work process and DB2 shadow process, and use them to establish the trace.