IBM SAP Technical Brief

Tuning SAP on DB2 for z/OS

Mark Gordon

IBM Solutions Advanced Technical Support

Version: 2.3
Date: March 30, 2010

1. 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 for z/OS database. The goal of the paper is that each can find a part of the material that is new and useful – an SAP BASIS administrator with experience on other databases will see some of the DB2 specific tuning tools and techniques, and DB2 DBAs with experience in traditional DB2 environments will see some SAP specific tools and techniques.

When doing performance monitoring for SAP on DB2 for z/OS, there are many different layers (SAP BASIS, SAP functional, DB, OS, network) and a variety of tools involved. Some problems can be solved in one layer, but some require monitoring and analysis in several layers. One of the goals of this paper is to show how to follow a problem through the various layers to the source. If different people monitor every layer, it is hard to have an integrated view of performance issues. While nobody can be an expert in all areas, if someone such as a DBA or BASIS administrator has an end-to-end view, they can call on an expert in a specific area, when a problem needs further investigation.

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 analysis of programs, determine where time is spent, and optimize these long running parts. This includes interpretation of STAT/STAD records, using ST05, SE30, SM50, SM51, SM66, etc. It will demonstrate how to drill-through 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.
- **To do a system health check**, review OS paging, CPU usage, and ST04 times (delay analysis in DB), SAP waits, 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. A health check should be done together with analysis of SQL.

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 under 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?”

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 is not important enough 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, systems programmer, SAP BASIS Administrator, etc.

2. Acknowledgements

2.1. Version 2

There have been many changes to and improvements in SAP's monitoring capabilities since the first version of this paper, as well as many changes in DB2. Thank you to Terry Purcell for his guidance and teachings on the workings of the DB2 optimizer, and for his suggestions on how to improve to the DB2 SQL analysis sections of this paper. Thank you to Johannes Schuetzner, for his advice on the use of the ever-improving SAP monitors and SAP management tools for DB2. Thank you to Joachim Rese and Georg Mayer for their advice on understanding and exploiting SAP's integration with DB2 for z/OS on BI systems.

2.2. Version 1

Thank you to Namik Hrle, who wrote and was the original instructor of the course "SAP R/3 on DB2 for OS/390 Performance Workshop", on which this white paper is based. My goal was to take his course materials, which were designed to be taught by an instructor, and make them into a paper showing the processes and tools for analyzing performance problems with SAP on a DB2 database for z/OS. I have added examples of solving specific problems, to demonstrate the process for starting from SAP performance indicators to drill-down to DB2 and z/OS indicators. Namik also reviewed this whitepaper, and offered many suggestions for improvements.

Several other people also provided valuable contributions to the paper. Thank you to Johannes Schuetzner, who advised me on the capabilities and usage of the new DB2 monitoring functions available with RFCOSCOL and Explain. Thank you to Lynn Nagel, Phil Hardy and Don Geissler who edited or reviewed the original version of this paper and made numerous suggestions for improvements. Thanks also to Mark Keimig, who showed me the process for SQL analysis with catalog statistics, which can be used with versions of SAP that do not have the enhanced explain described in note 535337. Thank you to Walter Orb, who showed me how to interpret symptoms of many SAP and AIX performance problems.

3. 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 hereof 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.

The paper contains examples from systems ranging from SAP R/3 Enterprise 4.7 with DB2 V7 up to ECC 6.0 with DB2 V8.

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 to clarify points for the paper, and do not necessarily reflect the way the example problems ran on the systems on which they were generated.

Sysplex performance issues, and z/OS 64-bit real performance issues are both out of the scope of this version of the document, and are discussed only briefly.

4. Trademarks

SAP®, R/3® , and SAP NetWeaver® are registered trademarks of SAP A.G. in Germany and in several other countries.
ABAP ™ is a trademark of SAP A.G. in Germany and in several other countries.
DB2®, MVS™, OS/390®, and z/OS® are trademarks or registered trademarks of IBM corporation.
Oracle® is a registered trademark of Oracle Corporation and/or its affiliates.

5. Feedback

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

6. Version Updates

- Version 1.0 – initial version
- Version 1.1 – new sections
  - DB procedure call information for APO systems
  - New explain functions with ST05 trace (Section 7.5.2)
  - SQLR example of merging STAT records and ST05 trace (Section 7.5.5)
  - Sample batch analysis with SE30 (Section 7.6.2)
  - ST04 enhancements with RFCOSCOL
  - Impact of parameter markers on DB2 (Section 8.3)
  - Updated packet loss problem example
  - Reference to RSTRC000 program (Section 13.1.20)
  - Updated section on RID processing with statement cache RID counters (Section 9.5.4)
- Version 2.0 – deleted old examples from 4.x systems, and added new sections based on ECC
  - Categories of problems (Section 7.3 and all subsections)
  - New transaction tracing examples (Sections 7.5.3 to 7.5.8)
  - REOPT(ONCE) as alternative to ABAP HINTs (Section 8.3.1)
  - New ST04 cache examples (Sections 8.5.2 to 8.5.5.1)
  - New lost packet network example (Section 9.3.1)
  - Rename document

- Version 2.1 – new sections
  - Missing time enqueue retry – Section 7.2.11.1
  - Identical selects – Section 7.4.8.5
- Version 2.2
  - Added DSRs in Section 7.3
- Version 2.3
  - Added modify program to match clustering sequence in Section 7.5.9.

7. Solving a performance problem for a specific program

7.1. Check STAT/STAD records for components of SAP elapsed time

Each time a program, transaction dialog step, or RFC finishes, a statistics record is saved by SAP. These statistics contain information about the components of elapsed time: CPU, database requests, etc. The response time breakdown for dialog steps can be viewed via the STAT or STAD transactions. The detailed statistics are periodically aggregated into the ST03 report. As described below in section 13.1.11, ST03 can be used to search for transactions or batch jobs that may need improvement. Since some of the detailed information in the STAT records is lost during ST03 aggregation, if a program is being investigated, the statistics records should be extracted for evaluation soon after the program runs.

STAD is an enhanced version of the STAT transaction. It contains support for DBPROC time (in APO systems) and also allows more flexibility in reporting time statistics.

If a performance problem has been reported for a program or transaction, one can use STAD data as a filter to examine performance, and build an action plan for doing more detailed analysis via traces, or other tools. STAD data shows symptoms of problems, and not causes.

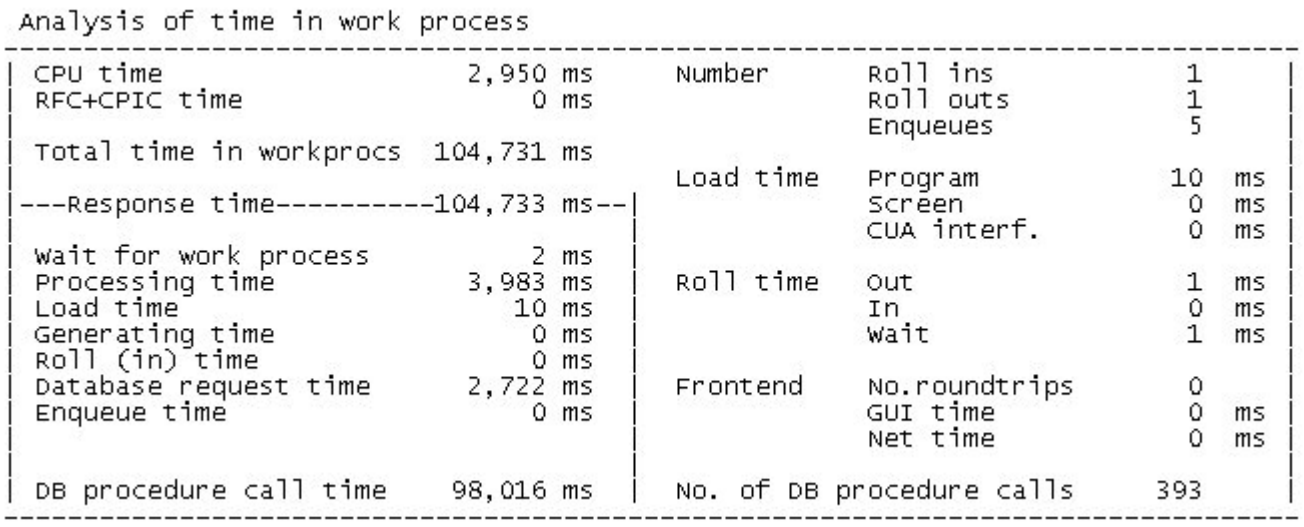
7.1.1. Description of STAT/STAD record time components

Early releases of SAP (3.1, 4.0, 4.5) may not have all the statistics categories shown on the following sample.

Analysis of time in work process

CPU time	50 ms	Number	Roll ins	1	
RFC+CPIC time	172 ms		Roll outs	1	
			Enqueues	0	
Total time in workprocs	923 ms				
		Load time	Program	1	ms
Response time	923 ms		Screen	0	ms
			CUA interf.	1	ms
Wait for work process	0 ms				
Processing time	883 ms	Roll time	Out	7	ms
Load time	2 ms		In	0	ms
Generating time	0 ms		Wait	0	ms
Roll (in+wait) time	0 ms				
Database request time	38 ms	Frontend	No.roundtrips	5	
Enqueue time	0 ms		GUI time	863	ms
			Net time	0	ms

Figure 1: Sample STAT record



Analysis of time in work process

CPU time	2,950 ms	Number	Roll ins	1	
RFC+CPIC time	0 ms		Roll outs	1	
			Enqueues	5	
Total time in workprocs	104,731 ms				
		Load time	Program	10	ms
---Response time----------	104,733 ms--		Screen	0	ms
			CUA interf.	0	ms
Wait for work process	2 ms				
Processing time	3,983 ms	Roll time	Out	1	ms
Load time	10 ms		In	0	ms
Generating time	0 ms		Wait	1	ms
Roll (in) time	0 ms				
Database request time	2,722 ms	Frontend	No.roundtrips	0	
Enqueue time	0 ms		GUI time	0	ms
			Net time	0	ms
DB procedure call time	98,016 ms	No. of DB procedure calls		393	

Figure 2: Sample STAD record

Following is a summary of the components of response time. See SAP note 8963 for a more detailed description for SAP releases up to 4.5B, SAP note 364625 for SAP 4.6 interpretation.

- **CPU time** is CPU time used on the application server. On long running batch jobs, this counter may wrap and be invalid. See SAP note 99584 for details.
- **RFC+CPIC** - time spent, as a client, waiting for RFC and CPIC calls.
- **Time in workprocs** is "*response time" – "Wait for work process*". The time the dialog step is queued waiting to be dispatched in a work process is not included.
- **Response time** elapsed time from start to end of dialog step
- **Wait for work process** is the time a dialog step was queued waiting to be dispatched in an SAP work process.
- **Processing time** is "*Response time" – "Database request time" – "Wait for work process" – "Roll (in+wait) time" – "Load time" – "Generating time*". One can think of it as "application is processing in the work process" time. "Processing time" is the time that SAP views the dialog step as being in a work process, and not waiting for data or programs required for execution. Since the counters for the component times used to calculate processing time can overflow on long jobs, and GUI RFC may or may not be included in Roll Wait, this indicator should be interpreted with care. See below for examples.
- **Load time** is time loading programs, screens, and CUA interfaces, which are individually broken out on the right of the STAT report.
- **Generating time** is time required to generate the executable version of program. If any of the components that make up a program have been changed since the last generation, then the program will be regenerated before execution.
- **Roll (in+wait) time**: an SAP context switch moves a dialog step into or out of a work process. This is called roll-in and roll-out. Roll wait is time spent when a dialog step is waiting for an RFC response, and rolled out to make the work process available to another dialog step.
    - Roll-in delay blocks a dialog step from running.
    - Roll-out does not block the dialog step from finishing, but it blocks another dialog step from using the work process.

    - Roll wait is a side effect of making an RFC call. Roll wait does not block subsequent dialog steps from using the work process. If it is high, one can examine the performance of the RFCs made by the dialog step. GUI RFC time is sometimes included, and sometimes not included in roll wait.
- **Database request time**: database request time includes three elements
    - time on the database server spent executing an SQL request
    - time on the application server spent processing requests for buffered tables
    - time spent on the network sending SQL to and getting replies from the database server.

  On long running batch jobs, the "Database request time" counter often overflows and is invalid, as can occur with CPU time above.
- **Enqueue** is time to process enqueues, which are SAP locks. Since an SAP Logical Unit of Work (LUW) may span several DB LUWs, SAP uses enqueues to serialize access to SAP objects, such as sales orders, customers, materials, etc.
- **DB Procedure** is time (on APO systems) to execute DBPROC calls to a Livecache DB server.
- **GUI time** is time to execute "GUI control RFCs" sent from application server to GUI.
- **Net time** is time sending GUI data across the network.

7.1.1.1. Description of STAT/STAD "missing time"

As described above, SAP gathers elapsed time information on many components of dialog step elapsed time. In some cases, you will note that the components of elapsed time do not account for all the elapsed time. The "processing time" field is a key in determining whether there is "missing time" that was not captured in the SAP STAT/STAD record. Processing time is *Response time – Database request time – Wait for work process – Roll (in+wait) time – Load time – Generating time*. The main components of elapsed time that are left in processing time are CPU time on the application server, enqueue time, and RFC/CPIC time (if not counted in roll wait).

On old releases where statistics counters can wrap, there can be delays that are not accounted for in the STAT/STAD records. When you are examining the STAT/STAD data for long running jobs, it is useful to compare processing time to the *sum of (CPU time + Enqueue time + RFC/CPIC time).* If processing time is much greater than this sum, it indicates that the dialog step occupied the work process, but was not doing activities in the SAP application layer. One of the following may be the cause:

- The statistics of some components (usually Database request time or CPU time) have wrapped, and the statistics are invalid. This happens with long running jobs and early versions of SAP (before 6.x kernel).
- There is operating system paging problem on the application server.
- There is a CPU overload on the application server.
- The program being executed is doing I/O to a file on the application server, e.g. an interface program that reads or writes UNIX files.
- The ABAP program is sorting a large internal table, and the sort has spilled over to sort on disk on the application server. This is just a variant of the previous problem with writing to an application server file, but is not under programmer control, but is done automatically by SAP.
- A batch job is using "Commit work and wait".

- A batch job is trying to acquire an enqueue for a locked object, failing to get the enqueue and retrying. (If a transaction cannot acquire an enqueue, it usually issues an error message to the user.)
- A job that creates and dispatches other jobs (e.g. a driver using RFC processing of IDOCs) is sleeping waiting for the end of the jobs it created.

7.1.1.2. Description of STAT/STAD detailed database request time

In addition to the time overview shown in Figure 1, one can display detailed database request information in STAT/STAD. Depending on your release of SAP, this stanza will look like Figure 3, where database time per request is reported, or Figure 4, where time per row is reported.

Analysis of ABAP/4 database requests (only explicitly by application)

Database requests total	279			Request time	30,956 ms		
				Matchcode time.	0 ms		
				Commit time	37 ms		
Requests on T??? tables	0			Request time	0 ms		

Type of ABAP/4 request	Requests	Database rows	Requests to buffer	Database calls	Request time(ms)	Avg.time per req.
Total	279	40,905	664	698	30,956	111.0
Direct read	128	19	108		234	1.8
Sequential read	126	40,864	556	676	30,393	241.2
Update	2	1		1	24	12.0
Delete	13	11		11	141	10.8
Insert	10	10		10	127	12.7

Figure 3: STAT database request with time per request

Analysis of ABAP/4 database requests (only explicitly by application)

Database requests total	147			Request time	388 ms		
				Matchcode time.	0 ms		
				Commit time	9 ms		

Type of ABAP/4 request	Database rows	Requests	Requests to buffer	Database calls	Request time (ms)	Avg.time / row (ms)
Total	144	147	95	36	388	2.7
Direct read	8	116	88		118	14.8
Sequential read	136	31	7	0	261	1.9
Update	0	0		0	0	0.0
Delete	0	0		0	0	0.0
Insert	0	0		0	0	0.0

Figure 4: STAT database request time with time per row

- Direct read is data read via the ABAP "SELECT SINGLE" call. This should be fully qualified primary index access. Direct reads may be returned from the SAP "single record" buffer on the application server. At the DB2 level, this will be a fetch, if DB2 is called.
- Sequential read is data read via the ABAP SELECT call. Sequential reads may be returned from the generic buffer on the application server. At the DB2 level, this will be a fetch, if DB2 is called.
- Update, insert, and delete correspond to DB2 update, insert, delete.

If the system reports time per request, since a sequential read request can return many rows, it is generally best to convert to time per row, in order to interpret sequential read performance. *If many calls are made which return no rows, the average per-row times may look high. Compare requests and rows, to check for this situation. Evaluate performance using the per-request times, if rows are seldom returned.* If many calls return no rows, that may be a sign that there are database requests against empty tables, or database requests which check for non-existent conditions. If empty tables are buffered in the application server "generic" buffer, it will reduce the performance impact of requests against the empty tables. If the table is and will remain empty, it may also be possible to modify the ABAP to remove the call to retrieve rows from the table.

In general, on an SAP system with a fast network to the database server, such as Gigabit Ethernet, SAP "direct read" times will be <2 ms per request, and SAP "sequential read" times will be <5 ms per request. If the application does lots of SAP "sequential read" array operations (look in STAT for database row count much greater than request count) then per-row sequential read times may be 1 ms or lower.

High per-call direct read times can be caused by:

- Lock contention on NRIV (since NRIV is selected with "select … for update")
- Low bufferpool hit rates or I/O contention on DB server
- Program error where "select single" is used incorrectly. (Select single should be fully indexed primary key, but an ABAPer can code select single for any select – ABAP does not know whether the select single is correctly indexed).

High per-row sequential read times may be caused by

- Inefficient SQL or bad access path used to access table (see section 0)
- Low bufferpool hit rates or I/O constraint on DB server

High per-row times for update and delete may be caused by

- Inefficient SQL or bad access path used to access table
- Application level locking contention (this is the most common cause)
- Low bufferpool hit rates or I/O constraint on DB server

High per-row insert times may be caused by

- Low bufferpool hit rates or I/O constraint on DB server
- DB2 page latch contention (in rare cases with very high insert rate)

System-wide DB server problems (CPU constraint, paging, network, etc) would cause all database calls to be slow.

7.1.1.3. Description of *stat/tabrec* and *rsdb/stattime* parameters

One can gather additional table access data by enabling the SAP profile parameter stat/tabrec and *rsdb/stattime*. *Stat/tabrec* records information in the STAT record about tables with the longest access time. Rsdb/stattime records table access times, which can be viewed in ST10. These parameters will increase CPU utilization on the application server, and are generally enabled for a short time so that one can determine which tables are causing delays in long running jobs. Since these STAT records are only available after a job finishes, one can review the STAT data and filter the problem based on the symptoms that are shown for the top tables after the problem jobs complete. Setting these parameters might be particularly useful when gathering initial performance data for jobs that run overnight, or when doing detailed workload analysis in a stress test.

After using stat/tabrec to find the tables causing the most delay, one can use ST05 to trace accesses to the table, or search the DBACOCKPIT (ST04) statement cache for the problem statement.

The following example is *stat/tabrec* data in STAT showing long change time on GLT0 – three seconds per change (6,130 ms / 2 updates). Performance on other tables such as CIF_IMOD, VBBE, and MSEG is ok, so the problem is not a system-wide problem, such as CPU constraint, paging, network, etc. We would need to investigate further to determine where the constraint is. The likeliest candidates for slow changes would be row locks, I/O bottleneck, or page latches.

Table accesses sorted by time (list might be incomplete)

Table name	Number of rows accessed: Total	Dir. reads	Seq. reads	Changes	Time (ms)
TOTAL	162	1	136	25	6,470
GLT0	2	0	0	2	6,130
TRFCQOUT	11	1	1	9	283
CIF_IMOD	135	0	135	0	32
VBBE	7	0	0	7	15
MSEG	7	0	0	7	10

Figure 5: stat/tabrec data

Stat/tabrec enabled STAT table times for an individual dialog step. In order to see table times for the entire SAP system, enable *rsdb/stattime* and use ST10 table statistics. Here, we see that overall update times for GLT0 are about 125 ms per update (4,452,028 ms / 35,035 updates). This is very high, and Figure 6 points to a pervasive problem with updates on this table. In normal circumstances, updates would take just a few ms each.

System: d660 Not buffered tables
Date & time of snapshot: 12/19/2001 22:47:36 System Startup: 12/19/2001 20:23:13

GLT0 G/L account master record transaction figures

Table description	Buffered	no
	Type	TRANSP
	Application class	FG
	Client dependent	yes
	Last modified	03.01.1999
	by	SAP

Operation Type	ABAP/IV Processor Requests	Fails	Database Calls Prepares	Opens	Fetch/Exec	Rows	Time [ms]
Select single	0	0	0	0	0		0
Select	0	0	0	0	0	0	0
Update	35,035	0	35		35,035	35,035	4,452,028
Delete	0	0	0		0	0	0
Insert	0	0	0		0	0	0
Buffer load			0	0	0	0	0

Figure 6: rsdb/stattime time statistics in ST10

Actions from STAT/STAD record analysis

This is the overall process to follow is to determine the major components of response time, evaluate whether they are candidates for improvement, and how they might be improved. Detailed examples of the activities listed in this section are contained in subsequent sections. This is a list showing how one might break down and approach a problem.

- If CPU time is low (e.g. less than 5-10% of elapsed time):
  - Check other response time components for delay
- If CPU time is high (e.g. over 70-80% of elapsed time):
  - Use SE30 to profile the transaction.
  - Look at routines with high time consumption as candidates for improvement.
- If CPIC+RFC time is high:
  - Trace the transaction with ST05 RFC trace.
  - Evaluate performance of RFCs to determine which RFC server is the source of the delay.
  - Go to that server and evaluate the performance of the RFCs as they are executed, to find source of delay in RFC code.
- If "wait for work process" time is high:
  - First look at this as a symptom of dialog steps staying too long in the work process, and look for problems in this or other concurrently running dialog steps for the cause.
  - If the performance of the concurrently running system workload OK, add more work processes.
- If the processing time is much greater than CPU time:
  - Check statistics and evaluate whether components might have wrapped. This may have happened on long running jobs.
  - If the stats have wrapped, and it is not clear what the real components of response time are, use the elapsed time analysis process in section 7.6.1.
  - Use ST06(N) (or OS tools) to check for CPU or I/O constraints on the application server.
  - Use SM50 or SQL trace (look for time gaps in the trace summary after commit work) to check for "commit work and wait"
  - Use ST05 enqueue trace to check for enqueue retries. SM12 statistics also show enqueue rejects, which occur when the requested object is already locked.
  - Use SM66 or SM50 to see whether the program is sleeping.
- If load time is high:
  - Check ST02 for swaps and database accesses on program, screen, and CUA buffers. Increase the size of buffer, if necessary
  - Check program logic to determine whether it uses "CALL TRANSACTION", in which case high load times are normal
- If generating time is high:
  - Check whether transports are being made frequently, or SAP data dictionary objects are being changed frequently.

- If roll (in+wait) is high:
  - Determine whether the problems is from roll-in or roll-wait by checking STAT for front-end and GUI times
  - If roll-in, use ST02 to check roll-area to determine if the amount used is greater than roll memory area
  - If roll-wait, examine the performance of GUI RFCs. See SAP note 51373 and 161053 for ways to minimize impact of GUI RFC calls.
- If DBPROC time is high:
  - Examine performance of the livecache DB server and the COM routines being called.
  - Examine performance of the network between APO systems and Livecache DB server.
- If database request time is high:
  - Evaluate time per request and time per row as discussed in section 7.1.1.2.
  - Gather additional information (via ST05 trace or *stat/tabrec* and *rsdb/stattime*) to determine where SQL is slow.
  - Check for inefficient SQL. See section 8.4 for examples of inefficient SQL.
  - Check for I/O constraints (using RMF III DEV, DELAY, etc) on active volumes and files.
  - Check bufferpool random hit-rates.
  - If change SQL is slow
    - Check the application to determine if changes are batched together into internal arrays in the program and performed just before commit, to reduce the time that row locks are held in DB2.
    - Check for lock contention. Lock contention can be confirmed with ST04 statement cache times, ST04 thread details, or DB2 trace with IFCID 44,45,226,227 (for SAP systems which do not have thread suspension times).
    - If application cannot be changed, or does not need to be changed, evaluate the impact of the lock suspensions
      - Lock suspensions in UP2 processes are not very important; UP2 is designed to de-couple statistics table updates from business table updates and process changes to statistics after changes to business tables.
      - Lock suspensions in UPD processes are more important, but not usually a critical problem, since UPD is asynchronous from user dialog processing.
      - Lock suspensions in DIA processes are most important, since the lock suspension is part of the end-user response time.
    - Evaluate controlling the level of parallelism in batch and update processes, to minimize lock contention. In systems with lock contention, there is generally an optimal level of parallelism for throughput, where fewer or more work processes give less throughput.
- If enqueue time is high
  - Calculate average time per enqueue.
  - If time per enqueue is good (1-3 ms), check application with ST05 enqueue trace to determine whether the application is making extraneous enqueues.
  - If time per enqueue is slow, check for enqueue constraints as shown in section 7.2.9.
- If GUI time is high
  - See SAP notes 161053 and 51373 regarding ways to improve performance of GUI RFC.

- If Net time is high
  - Examine the performance of the network between application servers and GUI.

7.2. Examples of problem indicators in STAT/STAD records

One important caveat when interpreting STAT statistics is that STAT data tends to be less than completely reliable. The counters may have missing time, or they may add up to more than the elapsed time. Time can be put in different categories, as when GUI time may or may not be included in Roll wait. When a performance problem has been reported for a program or transaction, don't use a single unusual STAT record to plan the investigation. Look for a pattern, to avoid wasting time working on a transient condition, or a problem in SAP statistics gathering. Use aggregated ST03 statistics for the transaction or program to confirm the "average" behavior of the program

7.2.1. High DB request time example

High DB request time is often a symptom of inefficient SQL or another DB performance problem.

If DB time is a very high percentage of elapsed time, consider checking for inefficient SQL. Here CPU time is only 1% of elapsed time and the DB request time is about 99%, which is generally a strong indicator of inefficient SQL. If the DB time is > 75%-80% of elapsed time, inefficient SQL is often the cause.

Analysis of time in work process

CPU time	203 ms	Number	Roll ins	2	
RFC+CPIC time	203 ms		Roll outs	0	
			Enqueues	0	
Total time in workprocs	20,201 ms				
		Load time	Program	5	ms
Response time	20,201 ms		Screen	0	ms
			CUA interf.	0	ms
Wait for work process	0 ms				
Processing time	325 ms	Roll time	Out	0	ms
Load time	5 ms		In	1	ms
Generating time	0 ms		Wait	0	ms
Roll (in+wait) time	1 ms				
Database request time	19,870 ms	Frontend	No.roundtrips	0	
Enqueue time	0 ms		GUI time	0	ms
			Net time	0	ms

Figure 7: STAT record with low CPU time

Use ST05 to trace and explain slow SQL statements. See section 7.3 and section 8.4 for examples of examining slow and inefficient SQL.

7.2.2. High CPU time example

If the dialog step spends most of its elapsed time using CPU on the application server, then one must look at where the time is spent.

Examine the program both when it is processing a few items and also many items, in order to search for issues in code scalability, such as inefficient access to internal tables.

In SAP, one can use the SE30 transaction to trace the program. See Section 7.6.2 for examples of running SE30. In the formatted ABAP trace, note which routines consume the most CPU, and examine the ABAP code for efficient programming.

If observations show much of the time is being spent in the SAP kernel, or operating system kernel, then open an OSS message. In this case, more detailed analysis of the SAP or OS kernels may be required.

In Figure 8, elapsed time is 586 seconds, with 505 seconds of CPU time. CPU time is 86% of elapsed time. This could be a sign of inefficient coding in the ABAP.

Analysis of time in work process

CPU time	505,516 ms	Number	Roll ins	2	
RFC+CPIC time	2,847 ms		Roll outs	0	
			Enqueues	252	
Total time in workprocs	586,419 ms				
		Load time	Program	90	ms
Response time	586,419 ms		Screen	1	ms
			CUA interf.	3	ms
Wait for work process	0 ms				
Processing time	536,773 ms	Roll time	Out	0	ms
Load time	94 ms		In	1	ms
Generating time	0 ms		Wait	0	ms
Roll (in+wait) time	1 ms				
Database request time	48,580 ms	Frontend	No.roundtrips	0	
Enqueue time	971 ms		GUI time	0	ms
			Net time	0	ms

Figure 8: STAT record with high CPU time

7.2.3. High RFC+CPIC time example

The detailed RFC data in the STAT record can help to determine whether the problem is specific to a system or an RFC. If a dialog step makes several calls to different RFCs, they are reported separately.

In addition to this historical RFC information, one can see the response times of RFCs during program execution by using the ST05 RFC trace. You must then go to the server for the slow RFC, and examine the cause of slow performance of the RFC.

```
 Remote function calls
--------------------------------------------------------------------------------
| Client subrecords                                                            |
| -----------------                                                            |
|                                                                              |
| Target          Z_TRILOGY_PRICER                                             |
| User ID         TEST01                                                       |
| Local  destin. ph0509_NST_20                    IP address 158.52.86.179     |
| Remote destin. (extern)                         IP address 10.10.10.2        |
| Program         SAPMV45A                                                     |
| Function        Pricing                                                      |
| Transaction ID                                                               |
| Received data                             489   Bytes                        |
| Sent data                                 969   Bytes                        |
| Calling time                            3,579   ms                           |
| Rem. exe. time                          3,562   ms                           |
--------------------------------------------------------------------------------
```

Figure 9: STAT RFC detail

7.2.4. Response time

High response time, in itself, is not a problem that can be fixed. Review the components of response time, to find out where the time is spent, and where the opportunities for improvement are.

7.2.5. Wait for work process example

Wait for work process means that SAP could not dispatch a dialog step in a work process, as all work processes were busy.

Wait for work process is often a symptom of another problem, where the root problem causes the dialog step to run slowly and keep the work process occupied for longer than optimal. This could be caused by CPU overload on the application server, OS paging on the application server, SAP buffer configuration, inefficient SQL, etc. Look at the components of response time to find where the time is spent, and work to improve performance in these areas.

If after looking for a root cause, none has been found, then add more work processes, or reduce the number of users on that application server. If the workload on a system grows, and additional work processes are not added to the SAP instances, “wait for work process” can result.

Analysis of time in work process

CPU time	150 ms	Number	Roll ins	1	
RFC+CPIC time	0 ms		Roll outs	1	
			Enqueues	1	
---Response time----------	1,093 ms				
Wait for work process	623 ms	Load time	Program	0	ms
Processing time	405 ms		Screen	0	ms
Load time	0 ms		CUA interf.	0	ms
Generating time	0 ms				
Roll (in+wait) time	65 ms	Roll time	Out	398	ms
Database request time	0 ms		In	65	ms
Enqueue time	0 ms		Wait	0	ms

Figure 10: STAT wait for work process – symptom of SAP roll area overflow

In Figure 10, note that wait for work process is about 60% of response time – 632 ms of 1093 ms. When we look for a root cause in the other indicators, this example shows that Roll-In and Roll-out are unusually large. With transactions, “roll in” and “roll out” are generally a few ms. In this example, a dialog step that should normally take about 150 ms (the CPU time) in a work process, occupied the work process for about 600 ms (65 + 150 + 398). See section 9.2.3 for an example of how to use ST02 to determine if the SAP roll-area is too small and causing the problem.

In Figure 11, the dialog step has some “wait for work process” time (2799 ms of 105,122 ms response time), but when checking the other components of response time, the database request time is the largest time component. Checking database request time, the insert time is very slow – 36 ms per inserted row. While wait time is not a large part of the elapsed time, this example shows again how wait time can be a symptom of another problem. If the database performance problem is solved (check for I/O constraints and DB2 lock/latch suspensions, etc) then the “wait for work process” time will likely go away, as the work processes (in general) will be occupied for a shorter time. (Improving DB performance of a dialog step does not help wait time for that dialog step, but reduces wait time for other dialog steps, which in the aggregate will reduce wait time.)

Analysis of time in work process

CPU time	5,740 ms	Number	Roll ins	0	
RFC+CPIC time	0 ms		Roll outs	0	
			Enqueues	0	
---Response time----------	105,122 ms--				
Wait for work process	2,779 ms	Load time	Program	6	ms
Processing time	4,337 ms		Screen	0	ms
Load time	6 ms		CUA interf.	0	ms
Generating time	0 ms				
Roll (in+wait) time	0 ms	Roll time	Out	0	ms
Database request time	98,000 ms		In	0	ms
Enqueue time	0 ms		Wait	0	ms

Analysis of ABAP/4 database requests (only explicitly by application)

Database requests total	3,344	Request time	98,000 ms
		Matchcode time.	0 ms
		Commit time	31 ms
Requests on T??? tables	0	Request time	0 ms

Type of ABAP/4 request	Requests	Database rows	Requests to buffer	Database calls	Request time(ms)	Avg.time per req.
Total	3,344	3,364	24	3,310	98,000	29.3
Direct read	12	1	11		11	0.9
Sequential read	7	62	13	9	145	20.7
Update	888	888		888	10,207	11.5
Delete	3	63		63	38	12.7
Insert	2,434	2,350		2,350	87,568	36.0

Note: Tables were saved in the tablebuffer.

Figure 11: STAT slow insert causes wait time

7.2.6. Processing time examples

High processing time is a symptom of a problem when it is not consistent with other statistics. In the simplest case, such as Figure 12, where there are no RFC and GUI calls, processing time should be very close to CPU time. In Figure 12, there is no "missing time", as can be seen by processing time and CPU time being nearly the same.

Analysis of time in work process

CPU time	7,281 ms	Number	Roll ins	26	
RFC+CPIC time	0 ms		Roll outs	26	
			Enqueues	50	
Total time in workprocs	17,577 ms				
		Load time	Program	13	ms
Response time	17,656 ms		Screen	0	ms
			CUA interf.	0	ms
Wait for work process	31 ms				
Processing time	7,252 ms	Roll time	Out	1801477	ms
Load time	13 ms		In	6	ms
Generating time	0 ms		Wait	48	ms
Roll (in+wait) time	54 ms				
Database request time	10,212 ms	Frontend	No.roundtrips	0	
Enqueue time	94 ms		GUI time	0	ms
			Net time	0	ms

Figure 12: STAT record with CPU corresponding to Processing time

In cases where there is a "missing time" problem, as described in section 7.1.1.1, processing time will be much larger than CPU time (plus RFC and enqueue if applicable). Note in Figure 13 that CPU time is 11,047 ms, while processing time is 91,747. In this case the dialog step was in the work process, but was not using SAP resources. This is a "missing time" indicator, and it will need to be evaluated while the program when it is running in order to determine the cause.

Analysis of time in work process

CPU time	11,047 ms	Number	Roll ins	2	
RFC+CPIC time	0 ms		Roll outs	0	
			Enqueues	0	
Total time in workprocs	174,629 ms				
		Load time	Program	6	ms
Response time	174,629 ms		Screen	1	ms
			CUA interf.	2	ms
Wait for work process	0 ms				
Processing time	91,747 ms	Roll time	Out	0	ms
Load time	9 ms		In	1	ms
Generating time	0 ms		Wait	0	ms
Roll (in+wait) time	1 ms				
Database request time	82,872 ms	Frontend	No.roundtrips	0	
Enqueue time	0 ms		GUI time	0	ms
			Net time	0	ms

Figure 13: Processing time shows missing time in SAP

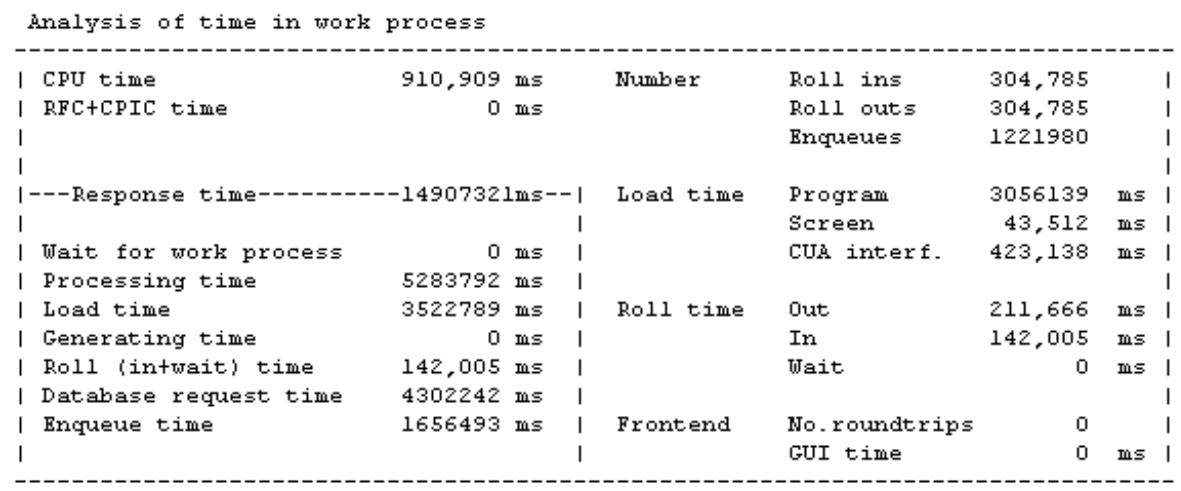
7.2.7. High load time example

Load time is generally a trivial percentage of response time. In the case of programs that call other programs (e.g. BDCs or custom programs using CALL TRANSACTION), it is normal to have load time be a high percentage of response time.

If the program buffer is too small and there are many swaps, then load time will affect programs when the executables are loaded.

In Figure 14, load time is about ¼ of response time – 3,522,789 of 14,907,321 ms. Database request time is also about ¼ of elapsed time.

In order to improve the performance, one might have to completely re-write the program, using, for example, a BAPI instead of CALL TRANSACTION. The effort of making the changes could outweigh the benefit from the improved performance.



Analysis of time in work process

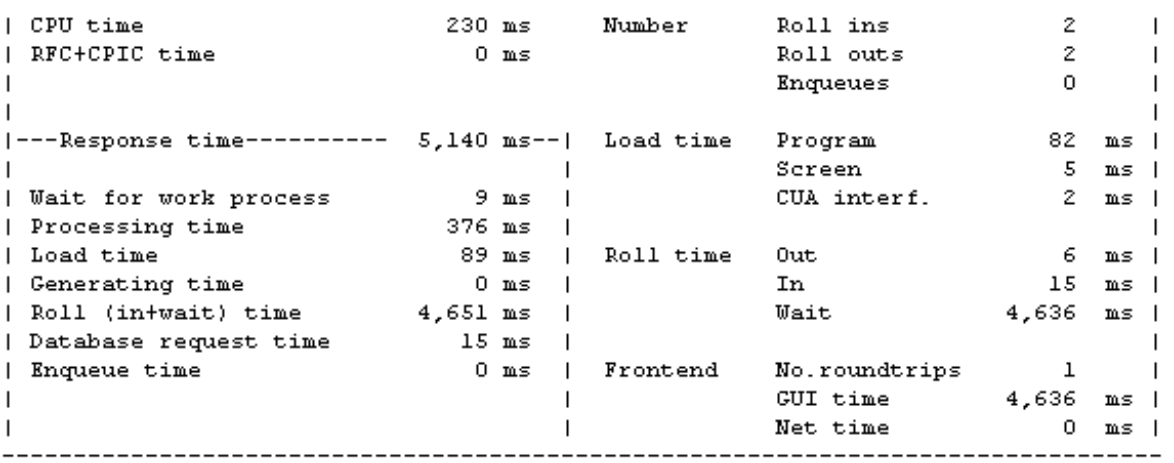
CPU time	910,909 ms	Number	Roll ins	304,785	
RFC+CPIC time	0 ms		Roll outs	304,785	
			Enqueues	1221980	
Response time	1490732 ms	Load time	Program	3056139	ms
			Screen	43,512	ms
Wait for work process	0 ms		CUA interf.	423,138	ms
Processing time	5283792 ms				
Load time	3522789 ms	Roll time	Out	211,666	ms
Generating time	0 ms		In	142,005	ms
Roll (in+wait) time	142,005 ms		Wait	0	ms
Database request time	4302242 ms				
Enqueue time	1656493 ms	Frontend	No.roundtrips	0	
			GUI time	0	ms

Figure 14: STAT high load time

7.2.8. Roll (in+wait) time example

Roll (in+wait) groups together two different kinds of wait. Roll-in delay is caused by a shortage memory-resident of roll area on the application server, and roll-wait is RFC wait on GUI calls. These categories are broken down on the right side of the statistics, under “Roll time”.

In Figure 15, roll (in+wait) shows that the performance issue for this dialog step is GUI time (roll-wait), not an application server ST02 ROLL area problem, which would be counted under roll-in time.



CPU time	230 ms	Number	Roll ins	2	
RFC+CPIC time	0 ms		Roll outs	2	
			Enqueues	0	
Response time	5,140 ms	Load time	Program	82	ms
			Screen	5	ms
Wait for work process	9 ms		CUA interf.	2	ms
Processing time	376 ms				
Load time	89 ms	Roll time	Out	6	ms
Generating time	0 ms		In	15	ms
Roll (in+wait) time	4,651 ms		Wait	4,636	ms
Database request time	15 ms				
Enqueue time	0 ms	Frontend	No.roundtrips	1	
			GUI time	4,636	ms
			Net time	0	ms

Figure 15: STAT roll (in+wait) GUI time

In Figure 16, roll in points to overflow of the memory roll area to disk on the application server. Use ST02 to follow up and review the roll area memory usage, as discussed in section 9.1.3.

Analysis of time in work process

CPU time	150 ms	Number	Roll ins	1	
RFC+CPIC time	0 ms		Roll outs	1	
			Enqueues	1	
---Response time----------	1,093 ms				
Wait for work process	623 ms	Load time	Program	0	ms
Processing time	405 ms		Screen	0	ms
Load time	0 ms		CUA interf.	0	ms
Generating time	0 ms				
Roll (in) time	65 ms	Roll time	Out	398	ms
Database request time	0 ms		In	65	ms
Enqueue time	0 ms		Wait	0	ms

Figure 16: STAT roll-in

7.2.9. Enqueue examples

Enqueue performance problems are generally seen only on very large systems, such as systems with hundreds of concurrent users, or many concurrent batch jobs.

Enqueue times should normally be very short. They are usually around one ms per enqueue. If enqueue processing is a significant percentage of time in the dialog step statistics, and the time per enqueue is high, there are three different causes:

- Central instance OS constraint (CPU or paging)
- Number of ENQ processes (a processor/threading constraint)
- I/O bottleneck on the ENQBCK file.

In the first case, where the central instance is overloaded from an OS level, other work processes on the CI (central instance) are using too much CPU or there is excessive paging, and the enqueue process cannot get enough CPU time to process requests.

Use ST06(N) (or programs such as vmstat) to check for CPU or paging problem.

The solution for this kind of problem is to move work processes off the central instance. For example, updates and UP2 work processes could be moved to other instances, batch processes could be defined as Class A, so that user jobs could not run there, and the SMLG definitions of login groups would direct users to login to other instances.

In the second case, where there is a processor/threading constraint, the ENQ process on the central instance is active all the time, and is constrained by the number of ENQ processes or constrained by the speed of the processors running ENQ.

The solution for this type of problem is to define more than one ENQ processes on the central instance, using the *rdisp/wp_no_enq* SAP parameter. Tests done by IBM have shown that performance of

enqueue increases for up to 3 enqueue processes on the CI. We have heard of systems with more enqueue processes defined, but do not have the performance data to recommend this configuration.

If a system with faster processors is available, running the central instance on this system will help alleviate an ENQ processor constraint.

The third type of ENQ performance problems is an I/O bottleneck on the ENQBCK file. At the end of an SAP transaction, ABAP programs issue an SAP "commit work" command. The "commit work" signals that the transaction is finished, and its update can be processed. Because SAP uses enqueues to serialize access to SAP objects, and these enqueues cannot be released until the update is complete, the "commit work" causes the ENQ process to write the state of the enqueues to disk. This ensures that the enqueues for the committed transaction will not be lost if the system crashes between "commit work" and update processing. This information is written to a file called ENQBCK, which is on the central instance. In situations where many "commit work" commands are being executed, the write activity to this file can become a bottleneck. See the SAP parameter *enque/backup_file* for the location.

The symptom of this problem is also long ENQ times. However, the ENQ process will generally not be running 100% of the time, and there will be very high I/O rates to the disk containing the ENQBCK file.

The solution to this problem is to place the ENQBCK file in a filesystem that resides on write-cached disk. In addition, it may be necessary to use a striped (either LV or disk striped) filesystem, to increase the I/O bandwidth of the ENQBCK file.

7.2.9.1. Enqueue processor constraint example

The symptom of this problem is long ENQ times seen in ST03 or STAT/STAD. Additionally, when one checks ST06(N) "top processes", one sees that the ENQ processes is using CPU nearly 100% of the time. SM50 will show processes in ENQ wait (when monitoring an SAP instance is not the central instance) or waiting on semaphore 26 (when monitoring the central instance). SM51 queue statistics on the Central Instance will show long ENQ request queues.

```
Mon Jul  3 12:52:58 2000
 interval  10   sec.
 Pid      Username     Command              CPU Util CPU Time   Resident Prior.
                                                [%]      [s]  size [kB]
-------------------------------------------------------------------------------
|76,228  |prdadm      |dw.sapPRD_DVEBMGS00 |100.11  |558:17  |  33,240  | 110  |
|111892  |prdadm      |dw.sapPRD_DVEBMGS00 | 75.23  |579:45  |  26,232  | 106  |
|107072  |prdadm      |dw.sapPRD_DVEBMGS00 | 38.16  |517:46  |  25,600  |  86  |
|69,076  |prdadm      |dw.sapPRD_DVEBMGS00 |  2.18  |  2:50  |  42,912  |  61  |
|98,834  |prdadm      |dw.sapPRD_DVEBMGS00 |  1.38  | 14:12  |  15,960  |  60  |
|106808  |prdadm      |ms.sapPRD_DVEBMGS00 |  0.99  | 11: 7  |   1,992  |  61  |
|10,078  |root        |mpci                |  0.99  |572:29  |  18,328  |  37  |
| 7,224  |root        |gil                 |  0.89  |658:35  |  18,372  |  37  |
|110934  |prdadm      |dw.sapPRD_DVEBMGS00 |  0.39  | 10:13  |  18,520  |  60  |
|34,896  |prdadm      |/usr/sap/PRD/SYS/ex |  0.19  |  2:44  |   1,504  |  60  |
|105856  |prdadm      |dw.sapPRD_DVEBMGS00 |  0.19  |  0:48  |  18,576  |  60  |
|19,092  |root        |/usr/lpp/adsm/bin/d |  0.09  |  0:21  |   1,332  |  60  |
| 9,038  |root        |log_kproc           |  0.09  |000:15  |  18,324  |  60  |
|18,848  |root        |/usr/sbin/nfsd 8    |  0.00  |179:16  |     200  |  60  |
| 9,854  |root        |/usr/sbin/syncd 60  |  0.00  |151:04  |     136  |  60  |
|23,996  |tracker     |/usr/lpp/tracker/bi |  0.00  |072:18  |     468  |  60  |
|18,404  |prdadm      |dw.sapPRD_DVEBMGS00 |  0.00  |071:47  |  32,564  |  60  |
```

Figure 17: ST06 > detail analysis > top CPU - showing processor constraint

Important note: Recent releases show CPU utilization in ST06 top processes adjusted by number of processors on the system. For example, on an 8-way system, 12% “CPU util” in ST06 “top processes” means that the work process is using 100% of one of the processors (100/8 = 12). The original way of reporting, where 100% meant 100% of a processor, was easier to interpret when looking for processor constraint problems.

To determine which reporting method is used, compare the CPU time used by a work process with the utilization reported over an interval, or use an OS level tool (such as tprof) that will display process utilization as a percent of a processor.

No.	Ty.	PID	Status	Reason	Start	Err	Sem	CPU	Time	Program	CliеUser	Action	Table
0	DIA	30112	running		Yes				82	SAPLBPT0	010 TRAIN001	Sequential read	BCONT
1	DIA	30222	running		Yes						010 SMITHAR	Roll In	
2	DIA	30560	running		Yes				81	SAPLEEWM	010 WF-BATCH	Sequential read	TEWOCODE
3	BTC	31582	stopped	ENQ	Yes				2135	SAPLSENA	010 BTCHUSER_ALL		
4	BTC	30892	stopped	ENQ	Yes				2147	SAPLSENA	010 BTCHUSER_ALL		
5	DIA	31024	waiting		Yes								
6	BTC	31984	stopped	ENQ	Yes		1		2144	SAPLSENA	010 BTCHUSER_ALL		
7	DIA	25810	running		Yes					RSMON000	010 GORDONMR		
8	DIA	25576	waiting		Yes								
9	DIA	24280	waiting		Yes								
10	DIA	19992	waiting		Yes								
11	BTC	18912	running		Yes				2142	SAPLE21A	010 BTCHUSER_ALL		
12	BTC	18616	stopped	ENQ	Yes				1896	SAPLSENA	010 BTCHUSER_ALL		
13	BTC	18116	stopped	ENQ	Yes				1894	SAPLSENA	010 BTCHUSER_ALL		
14	BTC	15244	stopped	ENQ	Yes				1891	SAPLSENA	010 BTCHUSER_ALL		
15	BTC	15024	running		Yes				360	ZCSVC0_M	010 BTCHUSER_ALL		
16	UPD	14736	waiting		Yes								
17	UPD	13994	waiting		Yes								
18	UPD	12412	waiting		Yes								
19	UPD	6314	waiting		Yes								

Figure 18: SM50 showing ENQ wait

In Figure 18, there are many processes showing “stopped ENQ”, which means waiting for enqueue. This instance is not the central instance. On the CI, enqueue wait is reported as semaphore wait. See Figure 22.

Request queue information

Application server sbspawl6_R21_06

Request type	Req.waiting	max req.wait	Max.req	Req. written	Req. read
NOWP	3	12	2,000	1,226,289	1,226,286
DIA	1	12	2,000	55,888	55,887
UPD	0	3	2,000	6	6
ENQ	1	288	2,000	1,111,234	1,111,233
BTC	0	2	2,000	14	14
SPO	0	2	2,000	618	618
UP2	0	2	2,000	1	1

Figure 19: SM51 > Goto > queue information - display of queues on SAP central instance

In Figure 19, the enqueue process was 288 requests behind at one point. While it is not unusual to see small queues on enqueue, if the queue gets to 50, 100, or above, one should look at the causes.

Since enqueue wait problems occur only at times of high enqueue activity, ST03 daily statistics will average them out and make them look less important than they are. Check periods of high activity, and look at the STAT records of jobs that run in those periods, to better evaluate the impact.

7.2.9.2. ENQBCK I/O constraint example

The problem reported is slow batch performance. ST03N shows that SAP lock time (enqueue time) is a large percentage of background and update time.

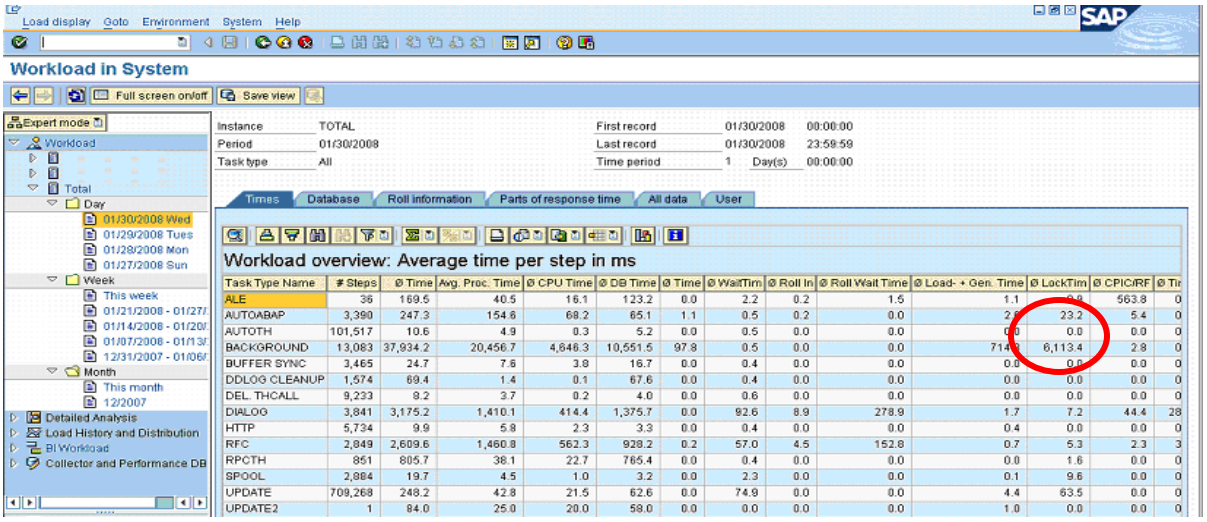


Figure 20: ST03N with high lock (SAP enqueue) time

We can also look at STAT records to evaluate the components of elapsed time. Following screen shots are same type of problem as in Figure 20, but from another system.

```
Server        : ge00565
Statistic file: /usr/sap/PRD/D00/data/stat
Analyzed time : 06/29/2000/18:00:00 - 06/29/2000/20:53:33   (with further selection criteria)
                                                 Response Memory   Wait      CPU      DB req.  Load/Gen kBytes   Phys. db
 End time Tcod   Program  T Scr. Wp User          time(ms) used(kB) time(ms) time(ms) time(ms) time(ms) transfer changes
-------------------------------------------------------------------------------------------------------------------------
|20:44:44        RM07II35 B      33 STO_PI_BATCH|5853955 |  6,346 |      0 |976,950 |1394897 |     21 |284478.2|  3,730 |
|20:48:00        RM07II35 B      32 STO_PI_BATCH|6051193 |  6,347 |      0 |1000320 |1411926 |     22 |291504.2|  3,888 |
|20:51:44        RM07II35 B      34 STO_PI_BATCH|6273886 |  6,346 |      0 |1037510 |1457464 |     22 |305425.9|  4,223 |
-------------------------------------------------------------------------------------------------------------------------

     Analysis of time in work process
    ------------------------------------------------------------------------------
    | CPU time                  1000320 ms     Number      Roll ins            2     |
    | RFC+CPIC time                   0 ms                 Roll outs           0     |
    |                                                      Enqueues        4,175     |
    |---Response time----------6051193 ms--|                                        |
    | Wait for work process           0 ms  |  Load time   Program            18  ms |
    | Processing time           1946000 ms  |              Screen              1  ms |
    | Load time                      22 ms  |              CUA interf.         3  ms |
    | Generating time                 0 ms  |                                        |
    | Roll (in+wait) time             9 ms  |  Roll time   Out                 0  ms |
    | Database request time     1411926 ms  |              In                  9  ms |
    | Enqueue time              2693236 ms  |              Wait                0  ms |
    ------------------------------------------------------------------------------
```

Figure 21: STAT long total and average enqueue times

In Figure 21, enqueue time is over 1/3 of the Response time, with an average enqueue call time of over 600 ms. This is very unusual. Enqueues should normally take just a few ms each.

Monitor the job while it is running, to look for the cause of the slow enqueues.

No.	Ty.	PID	Status	ReasonStart	Err	Sem	CPU	Time	Program	CliеUser	Action	Table
0	DIA	88764	running		Yes		26	22	SAPLSENT	010 STO_PI_BATCH		
1	DIA	79664	waiting		Yes							
2	DIA	74754	running		Yes				RSMON000	010 MGORDON		
3	DIA	70410	waiting		Yes							
4	DIA	59196	waiting		Yes							
5	DIA	27454	waiting		Yes							
6	BTC	91694	stopped	UPD	Yes			1209	RM07II35	010 STO_PI_BATCH		
7	BTC	113028	running		Yes			26268	ZRM07MAD	010 STO_ARCH	Delete	MSEG
8	BTC	44674	running		Yes			215	SAPLSAL2	010 STO_EXE		
9	BTC	57852	waiting		Yes							
10	DIA	96054	running		Yes		26	10		010 STO_OPC_CPIC		
11	DIA	85528	waiting		Yes							
12	BTC	56492	running		Yes		26	29460	SAPMM06E	010 DSMITH2		
13	DIA	66970	running		Yes		26	10		010 STO_OPC_CPIC		
14	DIA	50312	running		Yes		26	22	SAPLSENT	010 STO_PI_BATCH		
15	DIA	45422	waiting		Yes							
16	DIA	38466	waiting		Yes							
17	DIA	33566	waiting		Yes							

Figure 22: SM50 display on central instance showing Sem 26 (ENQ) wait

In Figure 22, there are many processes in enqueue wait. This is the central instance. On the CI, enqueue wait is displayed in SM50 as semaphore 26 wait.

Thu Jun 29 19:25:10 2000
interval 10 sec.

Pid	Username	Command	CPU Util [%]	CPU Time [s]	Resident size [kB]	Prior.
90,166	prdadm	dw.sapPRD_DVEBMGS00	26.69	58:34	40,720	78
10,078	root	mpci	2.38	429:11	18,328	37
107324	schedule	ksh /home/schedule/	2.08	23:24	596	60
13,938	root	dtgreet	2.08	27:46	1,624	61
7,224	root	gil	1.48	490:21	18,372	37
66,970	prdadm	dw.sapPRD_DVEBMGS00	0.99	69:46	30,932	60
44,674	prdadm	dw.sapPRD_DVEBMGS00	0.89	0:59	39,464	60
33,566	prdadm	dw.sapPRD_DVEBMGS00	0.89	14: 9	30,688	60
96,054	prdadm	dw.sapPRD_DVEBMGS00	0.79	68: 0	30,368	60
23,996	tracker	/usr/lpp/tracker/bi	0.79	53:37	464	60
88,764	prdadm	dw.sapPRD_DVEBMGS00	0.69	123:12	39,580	60
100422	prdadm	gwrd -dp pf=/usr/sa	0.69	54:53	3,576	60
59,196	prdadm	dw.sapPRD_DVEBMGS00	0.49	77:40	39,420	60
75,026	prdadm	/usr/sap/PRD/SYS/ex	0.39	3:19	1,740	60
74,754	prdadm	dw.sapPRD_DVEBMGS00	0.39	48: 1	30,380	60
50,312	prdadm	dw.sapPRD_DVEBMGS00	0.29	54:43	34,236	60
45,422	prdadm	dw.sapPRD_DVEBMGS00	0.29	42:37	33,232	60
42,918	prdadm	dw.sapPRD_DVEBMGS00	0.29	22:56	28,692	60
70,410	prdadm	dw.sapPRD_DVEBMGS00	0.29	65:44	40,744	60
85,528	prdadm	dw.sapPRD_DVEBMGS00	0.19	053:32	37,284	60

Figure 23: ST06 > detail analysis > top CPU - no processor constraint

But, unlike the previous example, the ST06 “top processes” shows there is no engine constraint on the enqueue process – the process using the most CPU is using 26% (of 100% possible in this release) of the time.

ST05 enqueue trace shows that some of the enqueue calls take a very long time, in particular after “COMMIT WORK”. In Figure 24, the DEQ call takes 389 ms, where it should normally take 1-2 ms.

```
|Transaction MM42|Work process no 26|Proc. Type  UPD|Client  500|User     PMCCAR|TransGUID 47A25E58329500BAE10080000A803019|Date  01/31/2008|
|HH:MM:SS.MS |Duration |Program |Obj. name |Op.    |Curs|Array|Recs.|RC    |Conn |Statement
|15:11:28.224|    2,781|RSM13000|          |EXECSTA|    |     |    0|     0|R/3  |COMMIT WORK ON CONNECTION 0
|15:11:28.227|  389,840|RSM13000|          |DEQ ALL|    |     |    0|     0|     |
```

Figure 24: ST05 ENQ trace

Since I/O on enqbck is another possible cause, check I/O activity with ST06.

```
Thu Jun 29 20:16:04 2000
 interval  10   sec.
 Disk        Resp. Util. Queue  Wait  Serv     Kbyte    Oper.
              [ms]   [%]  Len.  [ms]  [ms]      [/s]     [/s]
-------------------------------------------------------------
|  hdisk23 |  1  |  0  | N/A | N/A |  1  |       0  |     0  |
|      cd0 |  0  |  0  | N/A | N/A |  0  |       0  |     0  |
|   hdisk0 |  0  |  0  | N/A | N/A |  0  |       0  |     0  |
|   hdisk1 |  0  |  0  | N/A | N/A |  0  |       0  |     0  |
|  hdisk10 |  0  |  0  | N/A | N/A |  0  |       0  |     0  |
|  hdisk11 |  0  |  0  | N/A | N/A |  0  |       0  |     0  |
|  hdisk12 |  0  |  0  | N/A | N/A |  0  |       0  |     0  |
|  hdisk13 |  0  |  0  | N/A | N/A |  0  |       0  |     0  |
|  hdisk14 |  0  |  0  | N/A | N/A |  0  |       0  |     0  |
|  hdisk15 |  0  |  0  | N/A | N/A |  0  |       0  |     0  |
|  hdisk16 |  0  |  0  | N/A | N/A |  0  |       0  |     0  |
|  hdisk17 |  0  |  0  | N/A | N/A |  0  |       0  |     0  |
|  hdisk18 |  0  |  0  | N/A | N/A |  0  |       0  |     0  |
|  hdisk19 |  0  |  0  | N/A | N/A |  0  |       0  |     0  |
|   hdisk2 |  0  | 99  | N/A | N/A |  0  |      72  |   145  |
|  hdisk20 |  0  |  0  | N/A | N/A |  0  |       0  |     0  |
```

Figure 25: ST06 > detail analysis > disk - high I/O activity on UNIX disk

Note that hdisk2 is at 99% utilization. Now, check the LVs that are defined on the disk (lspv –l in AIX), or run a tool such as filemon, to confirm what is causing the activity on disk.

```
Cpu utilization:  3.8%

Most Active Files
------------------------------------------------------------------------
  #MBs  #opns   #rds   #wrs  file                 volume:inode
------------------------------------------------------------------------
   0.0      1      2      0  ksh.cat              /dev/hd2:8680
   0.0      1      2      0  cmdtrace.cat         /dev/hd2:8548
   0.0      1      0      1  wtmp                 /dev/hd9var:2072

Most Active Segments
------------------------------------------------------------------------
  #MBs  #rpgs  #wpgs  segid  segtype              volume:inode
------------------------------------------------------------------------
   8.4      0   2151   180a4c  page table
   0.0      0      4   e01c7  log

Most Active Logical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume               description
------------------------------------------------------------------------
  0.75      0  17208  905.9  /dev/lv01            /usr/sap/PRD
  0.03      0     32    1.7  /dev/loglv01         jfslog

Most Active Physical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume               description
------------------------------------------------------------------------
  0.73      0  17240  907.6  /dev/hdisk2          SSA Logical Disk Drive
```

Figure 26: filemon displays active filesystems

Since the files stanza of filemon generally does not report correctly on open files, use the LV stanza of the filemon report to find the active filesystem. Compare this to the SAP parameters controlling the location of the ENQBCK file.

For best performance, the ENQBCK file should be located on a disk with write-cache enabled, and in a filesystem with read caching.

If running enqueue server under USS on z/OS, ZFS should be used for the ENQBCK file, not HFS.

7.2.10. Frontend example

With GUI controls in a WAN environment the time to process RFC calls from the application server to the GUI can make a significant contribution to the elapsed time of a dialog step. On the other hand, the GUI control RFCs make it possible to reduce the number of screens on some transactions.

In 4.6, the ST03 workload summary DIALOG stanza includes average Frontend time.

When looking at a running system, SM50 or SM66 will show “stopped GUI” for each dialog step waiting for a GUI RFC. ST05 RFC trace can be used to trace the RFC calls to the GUI.

Analysis of time in work process

CPU time	219 ms	Number	Roll ins	2	
RFC+CPIC time	0 ms		Roll outs	2	
			Enqueues	0	
---Response time----------	4,059 ms	Load time	Program	0	ms
			Screen	4	ms
Wait for work process	0 ms		CUA interf.	2	ms
Processing time	3,414 ms				
Load time	6 ms	Roll time	Out	4	ms
Generating time	0 ms		In	2	ms
Roll (in+wait) time	545 ms		Wait	543	ms
Database request time	94 ms				
Enqueue time	0 ms	Frontend	No.roundtrips	3	
			GUI time	3,780	ms
			Net time	1,212	ms

Figure 27: STAT with long GUI time

In this example in Figure 27, note that GUI calls make up 3,780 ms of the 4,059 ms response time. Network data transfer time was 1,212 ms.

GUI time can be influenced by the speed of the frontend (PC), and by SAPGUI settings.

SAP notes 51373 and 161053 describe ways to optimize the performance of the GUI over a WAN. One can disable SAPGUI_PROGRESS_INDICATOR to reduce the number of calls to the GUI. Additionally, one can choose “classic gui”, or set the login for “low speed connection” in order to reduce the amount of communication between the presentation and application servers.

Net time is a function of the speed and latency of the network between the application server and frontend. If net time is slow, one must investigate it with network monitoring tools.

7.2.11. Missing time in STAT/STAD – suggested actions

- **The statistics of some component (usually Database request time or CPU time) have wrapped**, and the statistics are invalid. Monitor the running job using ST04 thread analysis, ST05, and SE30 to determine the components of elapsed time as described below in Batch Elapsed time analysis
- **There is operating system paging on the application server**. Use ST06(N) or an OS program such as vmstat to check for paging.
- **There is a CPU overload on the application server**. Use ST06(N) or OS program such as vmstat to check for CPU overload.
- **The program being executed is doing I/O to a file, e.g. an interface program that reads or writes UNIX files.** Use ST06(N) or OS program such as iostat or filemon to check or I/O activity.
- **The ABAP program is sorting a large internal table and the sort has spilled over to sort on disk.** Check location of *DIR_SORTTMP* in SAP parameters, and use ST06 or OS program such as iostat or filemon to check for I/O activity in this location.

- **A batch job is using "Commit work and wait".** When the program is running, watch it using SM50 or SM66. Check whether the job is often in the state "wait UPD", which means that it is waiting for "Commit work and wait" to complete. Check (via STAT records or ST05) that the updates are being processed efficiently. If so, investigate whether the program could be changed to do "commit work" so that updates are processed in parallel with the batch job. If the job must get a return code from "commit work and wait" in order to take error recovery action, then it would not be possible to change to use "commit work".
- **A batch job is trying to acquire an enqueue for an object locked by another process. It fails to get the enqueue, waits and tries again.** When the user program is running, check SM50 or SM66 and look for SAPLSENA in the program name. Use ST05 enqueue trace to confirm the problem. Look for repeated enqueues against the same object, where the return code (RC) is 2, which denotes that the enqueue could not be acquired. If these enqueues are being acquired as part of sales processing, check if OMJI (late exclusive material block) can be enabled. It will reduce this enqueue contention, an increase parallelism in sales processing, but also increases the load on the enqueue server on the central instance. When enabling OMJI, monitor the CI to confirm that it can support the increased load.
- **A batch job is sleeping, waiting for dispatched work to finish.** Use SM50 or SM66, and look for a status of SLEEP.

7.2.11.1. Missing time – enqueue reject and retry

As mentioned in Section 7.2.11, one of the causes of missing time is when ABAP programs wait to acquire an enqueue. This can be analyzed with either ST05, or SE30, or with both using ST12.

Analysis of time in work process

CPU time	37,890 ms	Number	Roll ins	0	
RFC+CPIC time	0 ms		Roll outs	1	
			Enqueues	138	
Total time in workprocs	101,848 ms				
		Load time	Program	39	ms
Response time	101,848 ms		Screen	0	ms
			CUA interf.	0	ms
Wait for work process	0 ms				
Processing time	88,583 ms	Roll time	Out	206	ms
Load time	39 ms		In	0	ms
Generating time	0 ms		Wait	0	ms
Roll (in+wait) time	0 ms				
Database request time	13,062 ms	Frontend	No.roundtrips	0	
Enqueue time	163 ms		GUI time	0	ms
			Net time	0	ms

Figure 28: Missing time stat record – enqueue retry example

In Figure 28, note that there is almost 40 ms that cannot be accounted for – the processing time is much higher than CPU time. If we trace the application with ST12 to gather SE30 ABAP trace and ST05 SQL and enqueue traces, we see that SE30 APAB trace shows over 30 seconds calling enqueue function, though the STAT record showed less than one second of enqueue time.

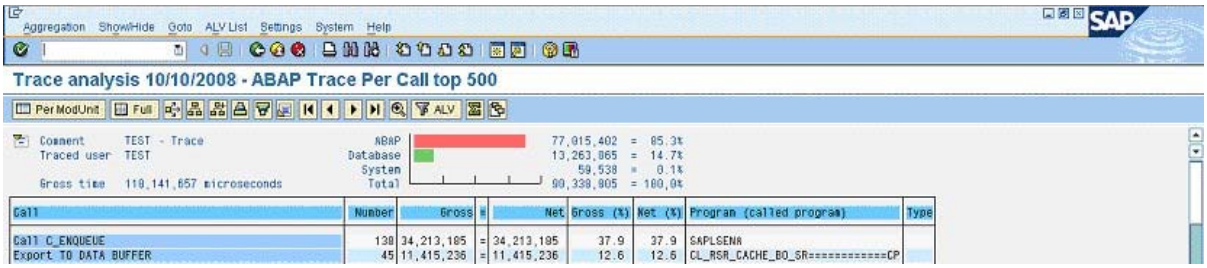


Figure 29: Missing time ABAP trace – enqueue retry example

If we compare the ABAP trace with the ST05 trace of SQL and enqueue, we see the reason for the difference. Each time the program fails to get the enqueue (RC=2), it waits for one second, then tries again. This time waiting is not externalized in any of the time categories of STAT, but is part of the missing time.

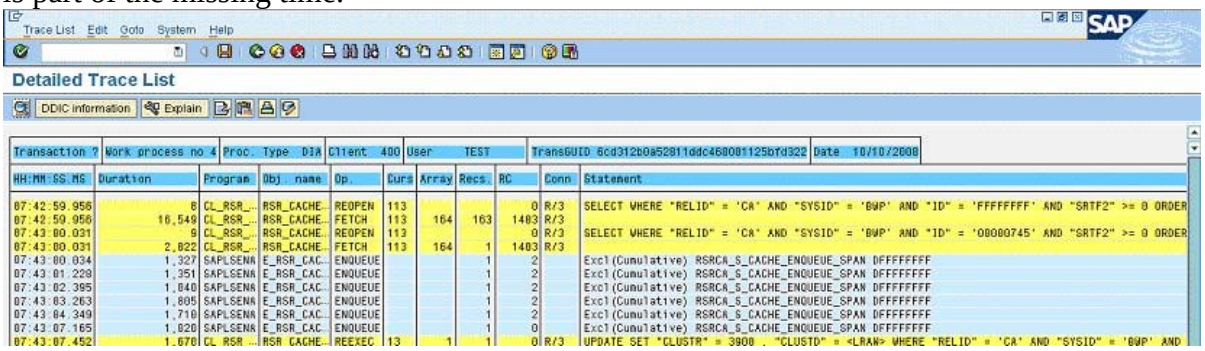


Figure 30: Missing time ST05 trace – enqueue retry

Wait for enqueue is an application issue. If the ABAP is SAP code, one would contact SAP to determine whether there are configuration changes that can be used to reduce the time that the enqueues are held, in order to reduce contention. If the ABAP is custom, then one would work with the developers on how to reduce the time the enqueues are held.

7.3. DSR statistics for Java Stack

DSR (Distributed Statistics Records) statistics are the Java analogue of ABAP STAT records – DSRs record the response time components for actions performed in the Java workload. Analysis of the DSR records is done in the ABAP stack, using either the STATTRACE (for individual DSRs) or ST03G (for aggregated DSRs). The configuration is described in the SAP "Monitoring Setup Guide".

As with STAT records, the DSR is used to evaluate the components of response time in an action, in order to determine which detailed traces might be needed to analyze performance. The DSRs are used in filtering problems, and do not generally point to a specific solution.

In the following example, a dual-stack nw04s system (BI with BI Java and EP) is used to demonstrate the integration of the DSR data with STAT records, to find out why a BI query runs slowly when run via the Java stack. We use the transaction STATTRACE, to evaluate the DSRs and STAT records.

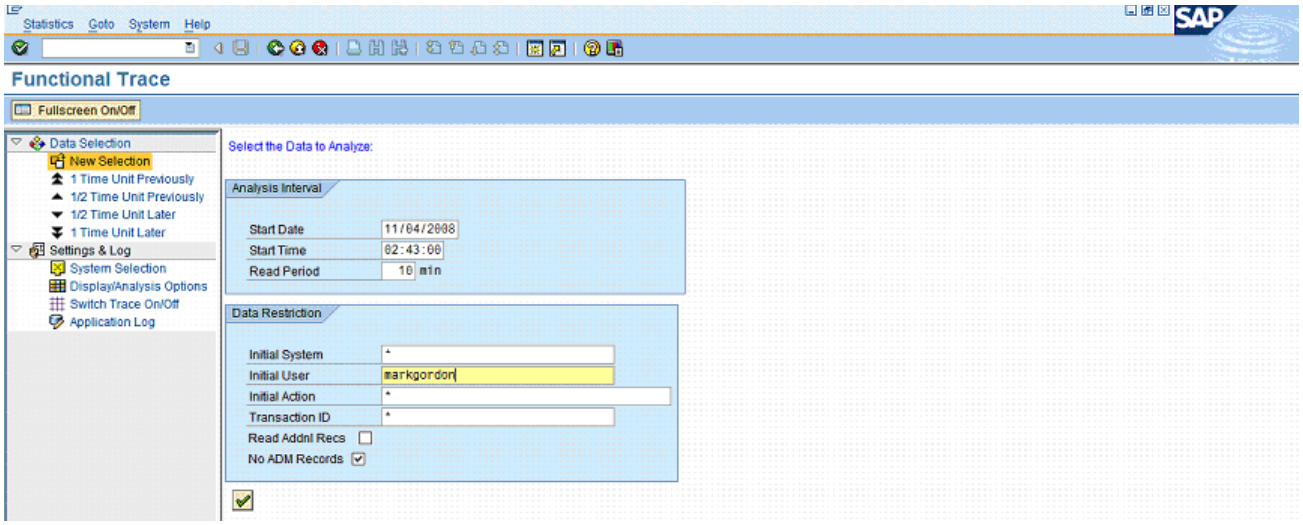


Figure 31: STATTRACE to select DSR and STAT records

Enter the time and other restrictions.

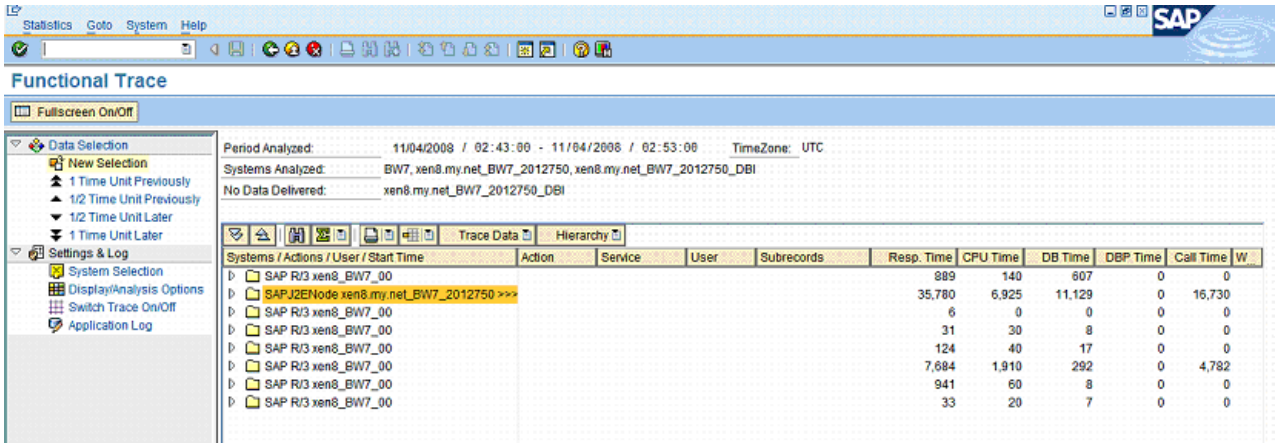


Figure 32: STATTRACE

In STATTRACE, DSR and STAT records with the same SAP Transaction ID are grouped together. Select one row, and press “Expand All” to see the details.

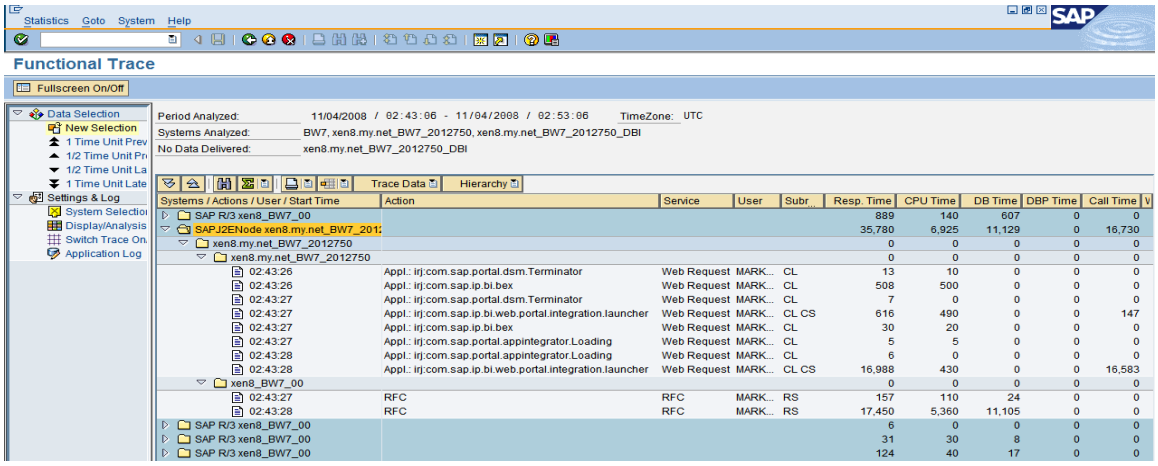


Figure 33: STATTRACE expanded

The response time column can give misleading information about response time in the summary line -- when the response time of concurrent actions are summed, the total response time will be longer than the elapsed time of all the actions. In this example, the longest DSR (16,988 ms) spends most of its time in RFC calls, so the RFC STAT records (e.g. l7,450 ms) are actually concurrently running in ABAP while the DSR's actions are active in Java. The timestamps show this. In the example in Figure 33, examine the DSRs to determine why the step runs over 16 seconds.

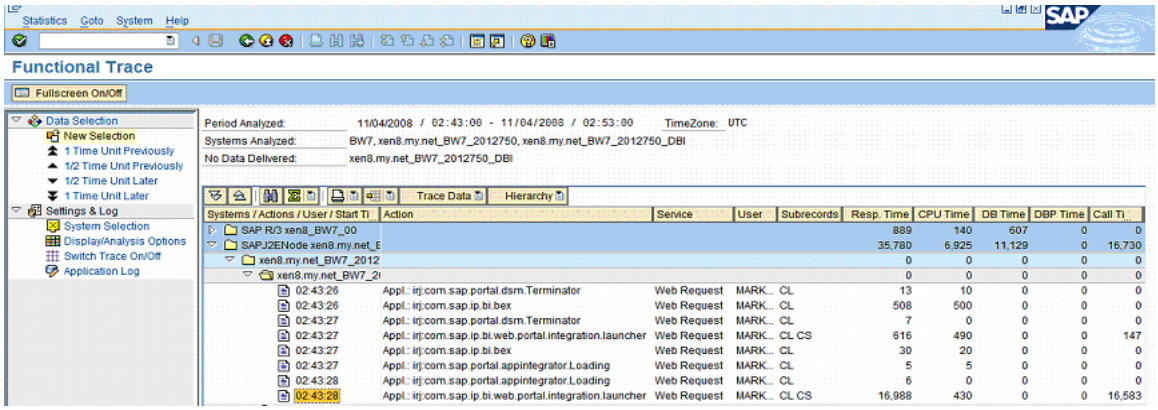


Figure 34: 16 second DSR

Drill in on the long DSR in Figure 34, and there are options for displaying the different components of response time.

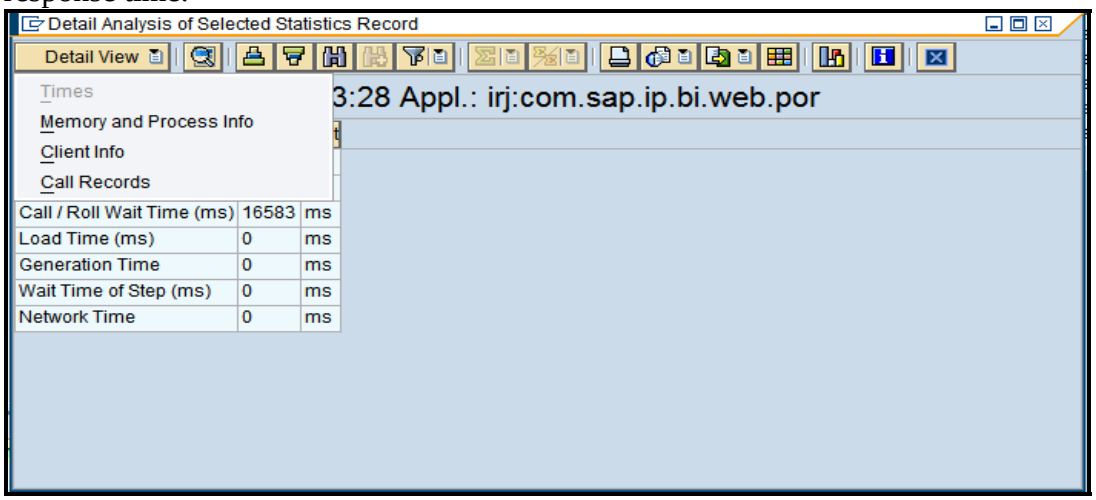
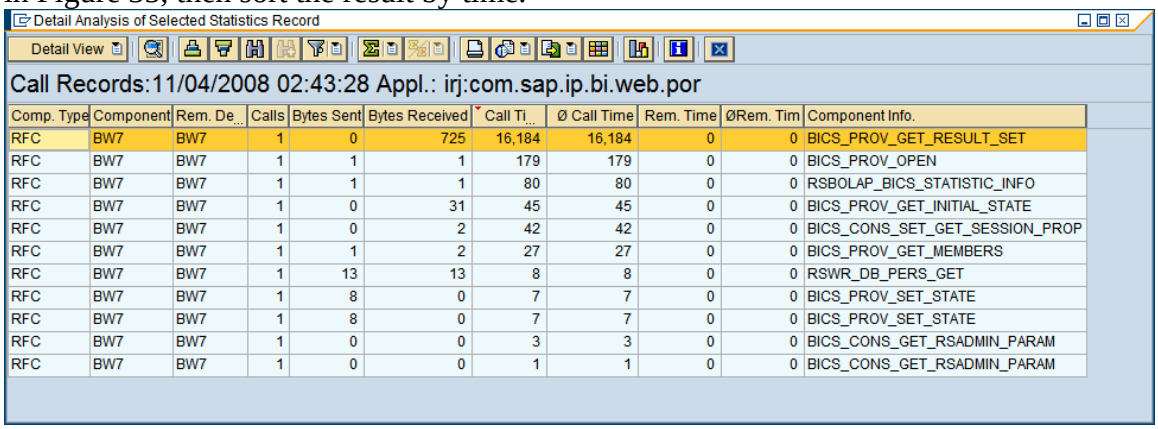


Figure 35: STATTRACE detail DSR analysis

Since Call time (RFC calls) is the longest response time component in Figure 34, select "Call Records" in Figure 35, then sort the result by time.



Comp. Type	Component	Rem. De...	Calls	Bytes Sent	Bytes Received	Call Ti...	Ø Call Time	Rem. Time	ØRem. Tim	Component Info.
RFC	BW7	BW7	1	0	725	16,184	16,184	0	0	BICS_PROV_GET_RESULT_SET
RFC	BW7	BW7	1	1	1	179	179	0	0	BICS_PROV_OPEN
RFC	BW7	BW7	1	1	1	80	80	0	0	RSBOLAP_BICS_STATISTIC_INFO
RFC	BW7	BW7	1	0	31	45	45	0	0	BICS_PROV_GET_INITIAL_STATE
RFC	BW7	BW7	1	0	2	42	42	0	0	BICS_CONS_SET_GET_SESSION_PROP
RFC	BW7	BW7	1	1	2	27	27	0	0	BICS_PROV_GET_MEMBERS
RFC	BW7	BW7	1	13	13	8	8	0	0	RSWR_DB_PERS_GET
RFC	BW7	BW7	1	8	0	7	7	0	0	BICS_PROV_SET_STATE
RFC	BW7	BW7	1	8	0	7	7	0	0	BICS_PROV_SET_STATE
RFC	BW7	BW7	1	0	0	3	3	0	0	BICS_CONS_GET_RSADMIN_PARAM
RFC	BW7	BW7	1	0	0	1	1	0	0	BICS_CONS_GET_RSADMIN_PARAM

Figure 36: STATTRACE Call Records

The long call time is caused by one client RFC call to the BW7 (ABAP stack) system. Now, we need to look at the STAT records for this Transaction ID, to see what causes the long response time in the BW7. We can also drill into the STAT record from the STATTRACE transaction.

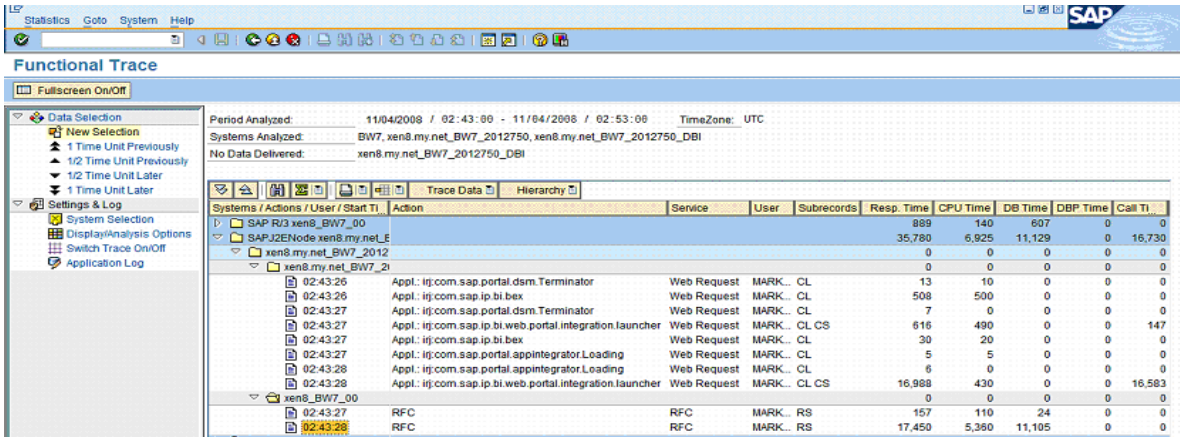


Figure 37: STATTRACE RFC STAT record

This is probably the STAT record (it is long enough), but we can check that it is the RFC call made from Java. Note that about 2/3 of response time is DB call time.

Drill into the timestamp, and the STAT record is displayed.

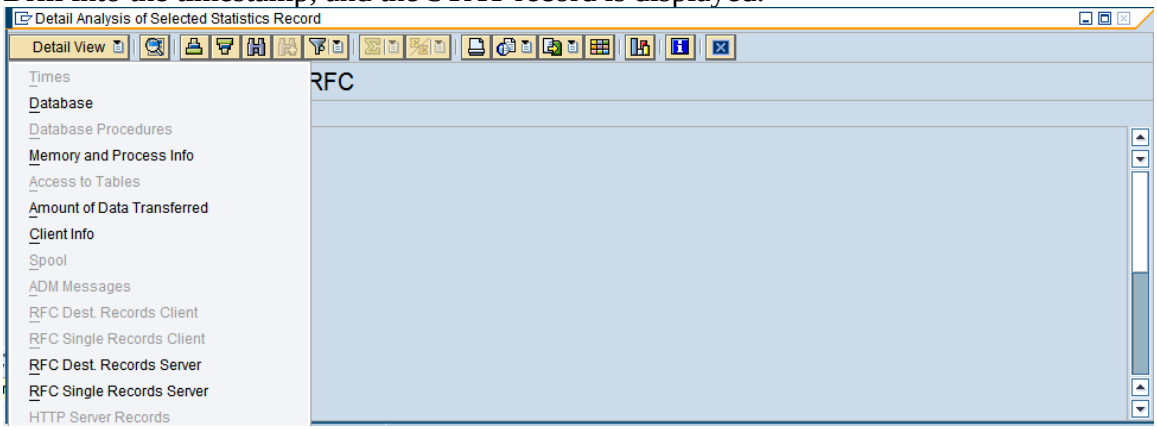


Figure 38: STATTRACE RFC STAT detail

Choose “RFC Single Records Server”, to see RFC calls for this STAT record. The BICS_PROV_GET_RESULT_SET server took 16 seconds.

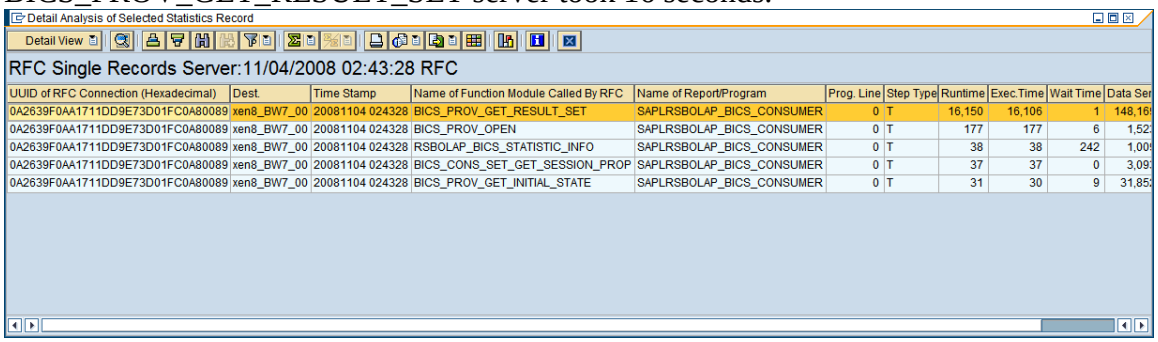


Figure 39: STATTRACE RFC Single Records

It is the same RFC call. STATTRACE uses Transaction Ids to group the steps, and you can also display the transaction ID for DSR or STAT record.

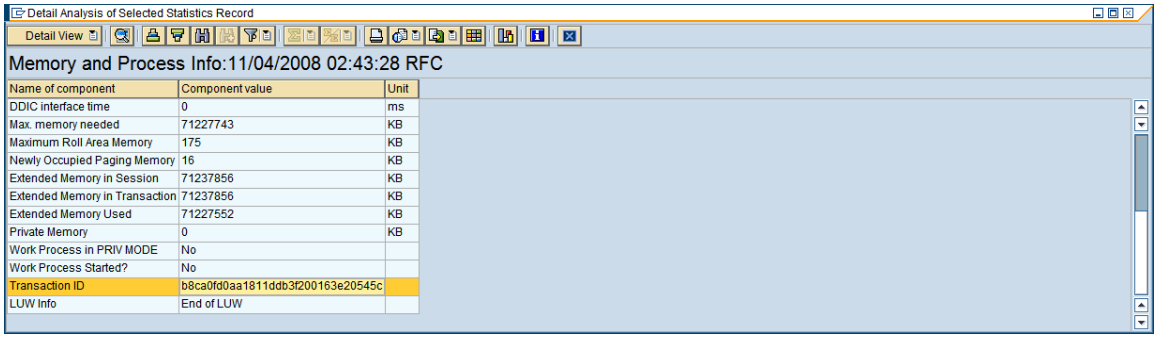


Figure 40: STATTRACE Transaction ID

The RFC calls in DSR and STAT can be traced with ST05 RFC trace.

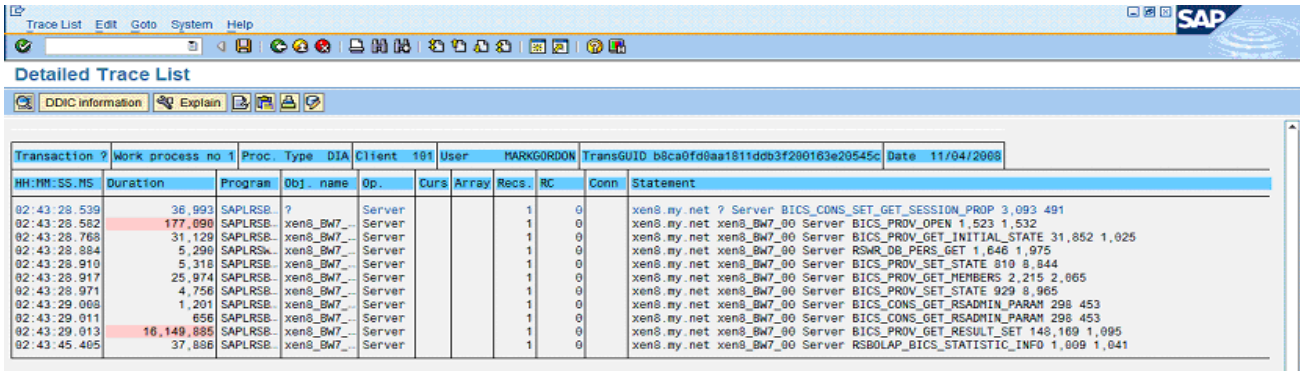
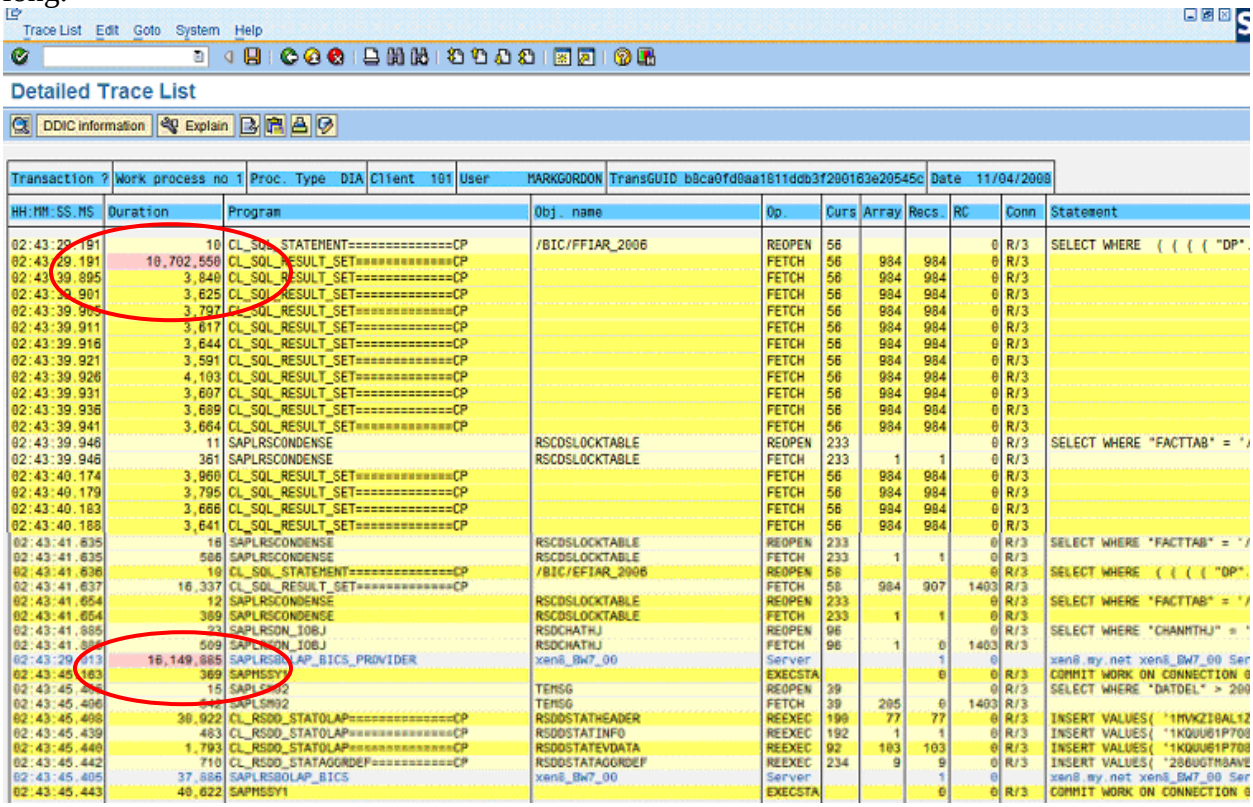


Figure 41: ST05 RFC trace matches RFC data in DSR and STAT

This trace was made during execution of the query that is being analyzed, and you can see that the RFC times on BW7 correspond to the RFC statistics in the DSR and STAT records.

So, from Figure 34 we know that the main component of time for the Java step is RFC call time to BW7. And we saw in Figure 37 that about 2/3 of the RFC time in BW7 was DB time. At this point, we can use ST05 to do an SQL trace of the query when it runs in BW7, to determine why the DB time is so long.


Trace List Edit Goto System Help

Detailed Trace List

DDIC information Explain

Transaction ? | Work process no 1 | Proc. Type DIA | Client 101 | User MARKGORDON | TransGUID b9ca0fd0aa1811ddb3f200163e20545c | Date 11/04/2009

HH:MM:SS.MS	Duration	Program	Obj. name	Op.	Curs	Array	Recs.	RC	Conn	Statement
02:43:29.191	10	CL_SQL_STATEMENT==========CP	/BIC/FFIAR_2006	REOPEN	56			0	R/3	SELECT WHERE (((("DP"
02:43:29.191	10,702,550	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:39.895	3,840	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:39.901	3,625	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:39.905	3,797	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:39.911	3,617	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:39.916	3,644	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:39.921	3,591	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:39.926	4,103	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:39.931	3,607	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:39.936	3,689	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:39.941	3,664	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:39.946	11	SAPLRSCONDENSE	RSCDSLOCKTABLE	REOPEN	233			0	R/3	SELECT WHERE "FACTTAB" = '/
02:43:39.946	361	SAPLRSCONDENSE	RSCDSLOCKTABLE	FETCH	233	1	1	0	R/3	
02:43:40.174	3,960	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:40.179	3,795	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:40.183	3,666	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:40.188	3,641	CL_SQL_RESULT_SET==========CP		FETCH	56	984	984	0	R/3	
02:43:41.635	16	SAPLRSCONDENSE	RSCDSLOCKTABLE	REOPEN	233			0	R/3	SELECT WHERE "FACTTAB" = '/
02:43:41.635	586	SAPLRSCONDENSE	RSCDSLOCKTABLE	FETCH	233	1	1	0	R/3	
02:43:41.636	10	CL_SQL_STATEMENT==========CP	/BIC/EFIAR_2006	REOPEN	58			0	R/3	SELECT WHERE (((("DP"
02:43:41.637	16,337	CL_SQL_RESULT_SET==========CP		FETCH	58	984	907	1403	R/3	
02:43:41.654	12	SAPLRSCONDENSE	RSCDSLOCKTABLE	REOPEN	233			0	R/3	SELECT WHERE "FACTTAB" = '/
02:43:41.654	389	SAPLRSCONDENSE	RSCDSLOCKTABLE	FETCH	233	1	1	0	R/3	
02:43:41.885	33	SAPLRSDN_IOBJ	RSDCHATHJ	REOPEN	96			0	R/3	SELECT WHERE "CHANMTHJ" = '
02:43:41.886	509	SAPLRSDN_IOBJ	RSDCHATHJ	FETCH	98	1	0	1403	R/3	
02:43:29.413	16,149,885	SAPLRSBOLAP_BICS_PROVIDER	xen8_BW7_00	Server			1	0		xen8.my.net xen8_BW7_00 Ser
02:43:45.163	369	SAPMSSY1		EXECSTA			0	0	R/3	COMMIT WORK ON CONNECTION 0
02:43:45.[illegible]	15	SAPLSM02	TEMSG	REOPEN	39			0	R/3	SELECT WHERE "DATDEL" > 200
02:43:45.406	[illegible]	SAPLSM02	TEMSG	FETCH	39	205	0	1403	R/3	
02:43:45.408	30,922	CL_RSDD_STATOLAP==========CP	RSDDSTATHEADER	REEXEC	190	77	77	0	R/3	INSERT VALUES('1MVKZI0AL1Z
02:43:45.439	463	CL_RSDD_STATOLAP==========CP	RSDDSTATINFO	REEXEC	192	1	1	0	R/3	INSERT VALUES('1KQUU61P700
02:43:45.440	1,793	CL_RSDD_STATOLAP==========CP	RSDDSTATEVDATA	REEXEC	92	103	103	0	R/3	INSERT VALUES('1KQUU61P700
02:43:45.442	710	CL_RSDD_STATAGGRDEF==========CP	RSDDSTATAGGRDEF	REEXEC	234	9	9	0	R/3	INSERT VALUES('286UGTM8AVE
02:43:45.405	37,886	SAPLRSBOLAP_BICS	xen8_BW7_00	Server			1	0		xen8.my.net xen8_BW7_00 Ser
02:43:45.443	40,622	SAPMSSY1		EXECSTA			0	0	R/3	COMMIT WORK ON CONNECTION 0

Figure 42: ST05 SQL and RFC trace of query

The trace in Figure 42 shows that there is one SQL statement, a select on /BIC/FFIAR_2006 that makes up 10 seconds of DB time in the 16 second RFC. We would then examine this (using techniques discussed below) to optimize it.

7.4. Types of problems causing high DB request time

7.4.1. Indexes do not support predicates

Local predicates are the "column operator value" clauses in SQL. In order to execute these efficiently, the column must be in an index. If the column is 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 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 in the join predicates.

There is an example of this in Section 7.5.3.

7.4.2. 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, so if a program has the value of an order number and wants to retrieve the associated delivery, the program could use either table. But, if the program selects from the delivery table using order number, there is no index for order number and the select is slow. If the program selects from the order table, 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 8.5.3.

7.4.3. 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

```
LOOP AT stpox_tab_tmp INTO stpox_tab.
  SELECT SINGLE j_3amind FROM mara INTO stpox_tab-j_3afs_flg
         WHERE  matnr = stpox_tab-idnrk.
  APPEND stpox_tab.
ENDLOOP.
```

Figure 43: LOOP with select

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 SAP will create an IN list (there is one column=value in the WHERE) or an OR (if there is more than one column=value clause in the WHERE.

7.4.4. Data skew causes wrong access path to be chosen

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 assumes that each value specified for AUFNR will return 10 rows.

However, if many rows contain a single value, then the data is skewed -- that is the distribution of values in rows is not uniform. If data is skewed, DB2 requires additional RUNSTATS FREQVAL statistics, combined with either ABAP HINTs or REOPT(ONCE) BIND option, in order to use execution values to determine the skew.

There is an example of this in Section 8.3

7.4.5. REOPT(ONCE) impact

The default method of preparing statements with parameter markers can cause access path problems in some situations, as shown in the example in section 8.3. To address, the REOPT(ONCE) option was created, which allows a statement to be prepared with values, but still shared by all dialog steps that execute the SQL, regardless of the runtime parameters. However, if the values used for the first execution of the statement are not representative of the normal execution, this can cause DB2 to choose the wrong access path.

This is described in Section 8.5.2

7.4.6. Impact of index column order with range predicates

If an index has n columns, such as (column1, column2, column3) and an SQL statement has local predicates column1 = value1 and column2 = value2 and column3 GT value3, then DB2 can use index matching access on all three columns in the index.

If the index columns were ordered (column1, column3, column2) then DB2 could only do index matching access on two columns, since any index columns to the right of a range predicate can't be used for index matching access. They can, however, be used for index screening, which is less efficient.

When creating custom indexes, place columns that will have equal predicates (column = value) as the first columns, and place columns that will be used for range predicates (gt, lt, between) as the later columns.

There is an example of this issue in Section 8.5.5.

7.4.7. Table is not clustered to support key business processes

When a table is clustered by an index, rows in the table are in the same order as the index. This means that rows that are close together in the index will be close together (if REORGS are done) in the table. When doing array fetch operations, where many rows are retrieved in one call to the database, clustering the rows will reduce I/O delay, and increase the speed of the SQL.

Since there can only be one clustering index, if you change the clustering index from the default delivered by SAP, they you are choosing to optimize access via a different access than usual. This will help performance of programs that use clustering index, but slow performance of programs that read the table using the former clustering index.

An example of changing clustering sequence is in Section 7.4.7.

7.4.8. Unnecessary SQL

7.4.8.1. 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 7.5.7

7.4.8.2. 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 7.5.6.

7.4.8.3. 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 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 7.5.5.

7.4.8.4. Program retrieves rows that are not needed

There is an example of this problem in Section 7.5.4

7.4.8.5. Duplicate (or Identical) selects

SAP ST05 traces can be analyzed for duplicate selects, where the same rows are repeatedly read. This is an application design issue, and cannot be fixed with DB optimization. Check the trace to determine whether the analysis contains multiple commit points. If there are multiple units of work, then it would be reasonable for each UOW to retrieve the current values from the tables. If the trace contains a single UOW, and there are many duplicate selects, then it may be possible to change the structure of the program to retrieve the data once.

Depending on the SAP version, ST05 calls this either “Duplicate selects” or “Identical selects”.

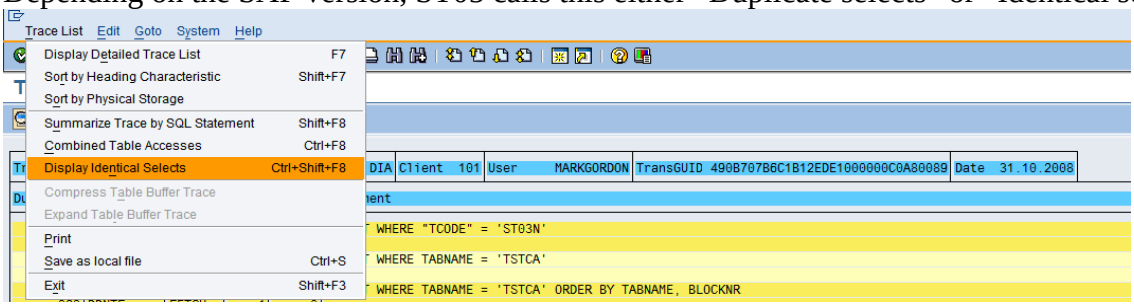


Figure 44: ST05 Display identical selects

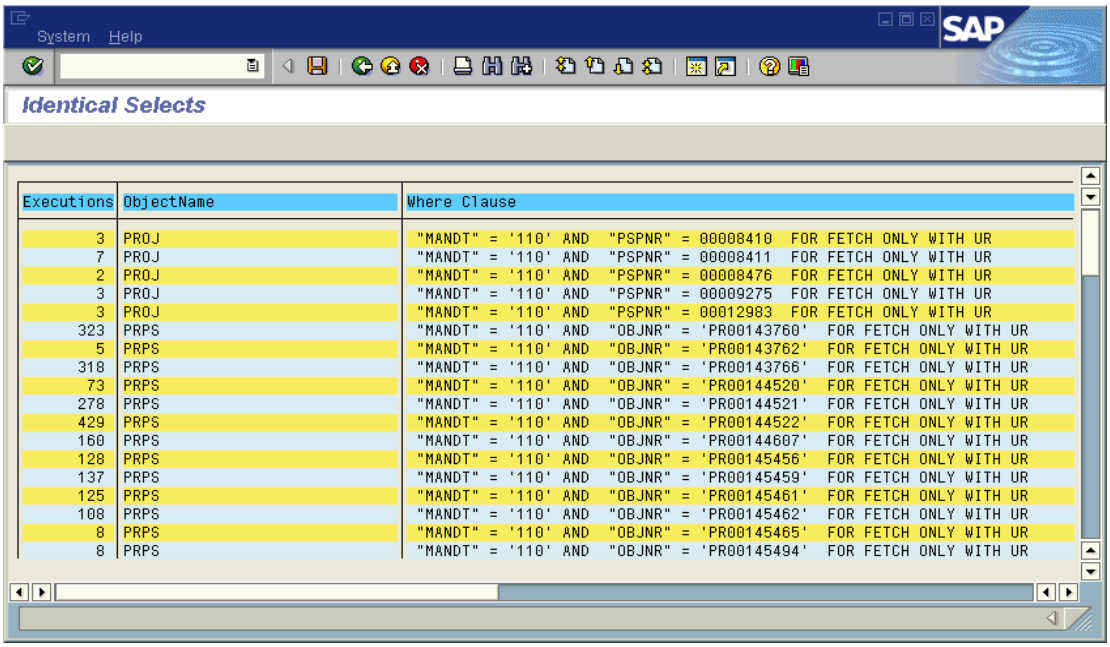


Figure 45: ST05 Identical selects analysis

7.5. Transaction

7.5.1. Analysis process for transactions and batch jobs

In the case where there is a performance problem with a specific program or transaction, there are two different paths to take, depending on where the transaction spends its time. If most of the time is spent in CPU on the application server, use SE30 to profile the transaction, and determine where the time is spent. Otherwise, start with ST05, which can be used to evaluate database calls, enqueue activity, and RFC activity.

7.5.2. Guidelines for making database changes

In general, we use this matrix of inputs, when deciding how to address SQL performance problems.

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

ABAP Creator	Table Creator	Action when predicates do not match indexes
SAP	SAP	• Check data dictionary for SAP indexes which are defined in DDIC but not active in DB • Check OSS notes • Open OSS message to SAP
Custom	Custom	• Rewrite program if possible • Evaluate table access patterns, whether table can be buffered • Create index

Custom	SAP	• Check data dictionary for SAP indexes that are defined in DDIC but not active in DB • Review use of data model (will generally fix problem) • Change program • Create index (should very seldom be necessary)

7.5.3. ZCR3 – indexes do not match local predicates

This is an example of a program where the local predicates do not match the available indexes. This type of problem is described in Section 7.3.

Performance problem has been reported on transaction ZCR3. Reviewing ST03N, we see that that the majority of CPU time is DB request time.

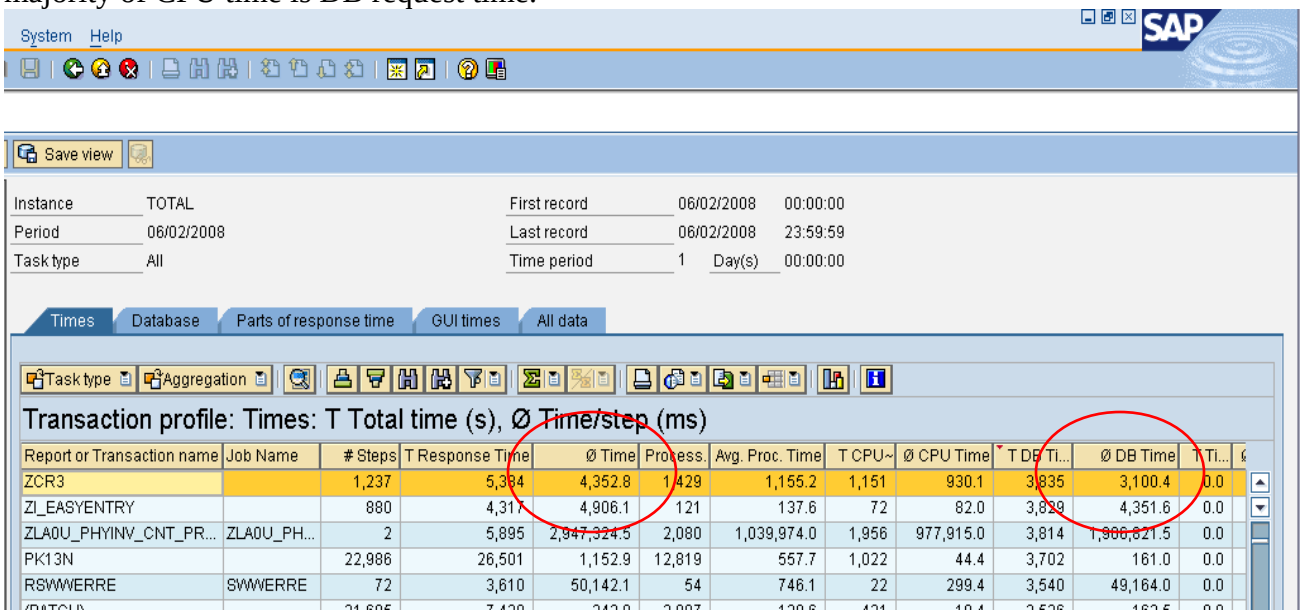


Figure 46: ZCR3 ST03N time components

The average response time is about four seconds and the ratio of DB to CPU is about 3/1, which does not seem to indicate a big DB performance problem. We can look at the detailed statistics records to see whether the averages reflect the transaction performance.

We can also check the statistics records with the STAD transaction, where we see the response time components for individual dialog steps. Note that in Figure 47, the DB request time is about seven times as large as the CPU time, which often is a symptom of a problem in the SQL or database. Since some of the dialog steps are very fast, the long running dialog steps were not visible the averages.

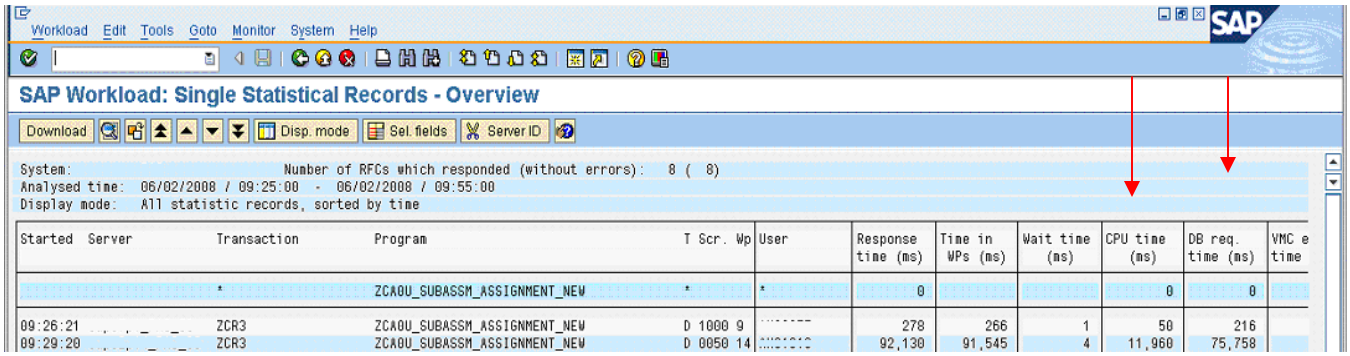


Figure 47: ZCR3 statistics records

Run ST05, then press “activate trace with filter”. Enter the filters (username or transaction) for the trace. ST05 traces on a single SAP application server instance.

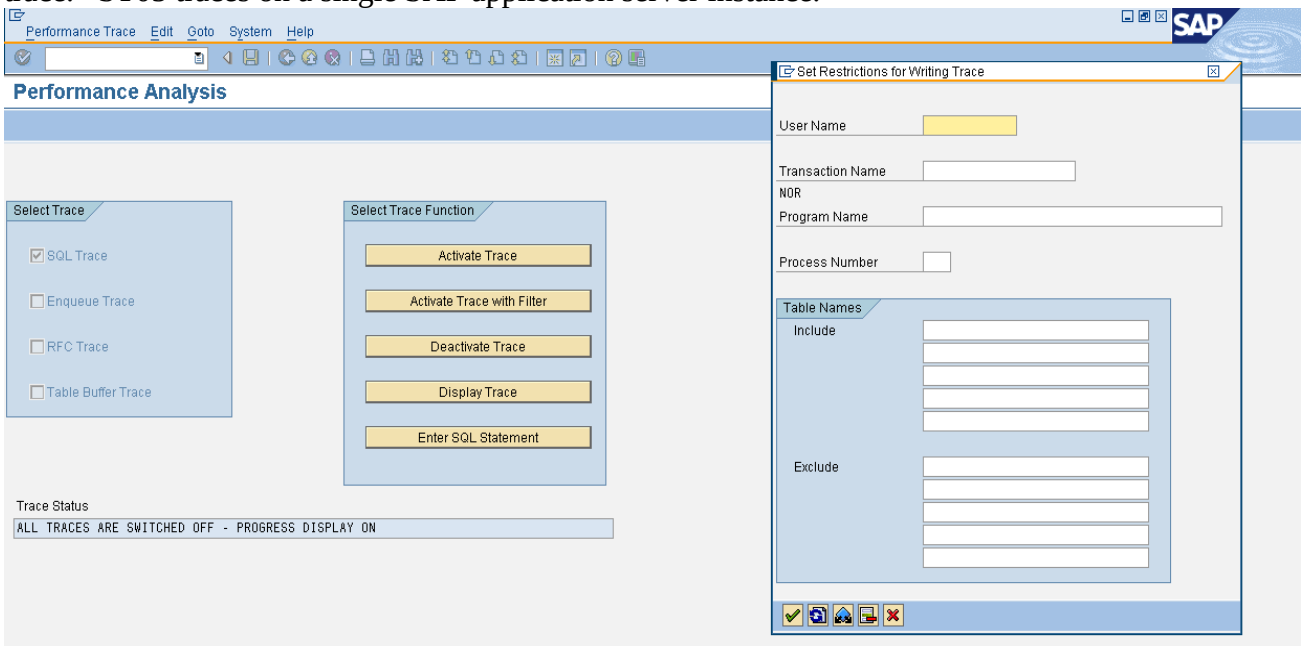


Figure 48: ST05 trace with filter

After the trace is done, run ST05 – “deactivate trace”, and then “display trace”. In Figure 49, the Duration is microseconds, and the “Recs” column on the FETCH line shows rows returned from DB2. This example is 60 ms for 0 rows, which is slow. Fetching a row (or determining that no rows qualify) usually takes 1-2 ms.

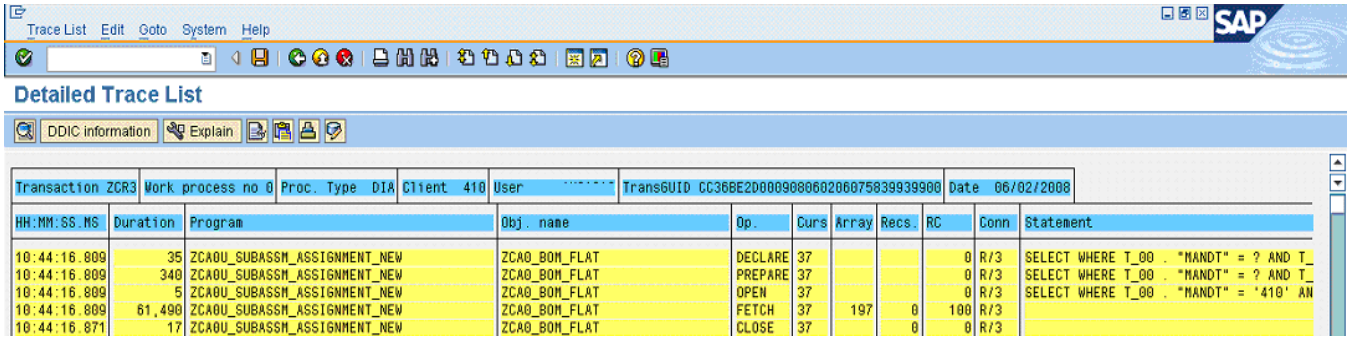


Figure 49: ZCR3 ST05 trace

Click “Trace List > Summarize trace by SQL statement” to group SQL, and sum the time for each statement.

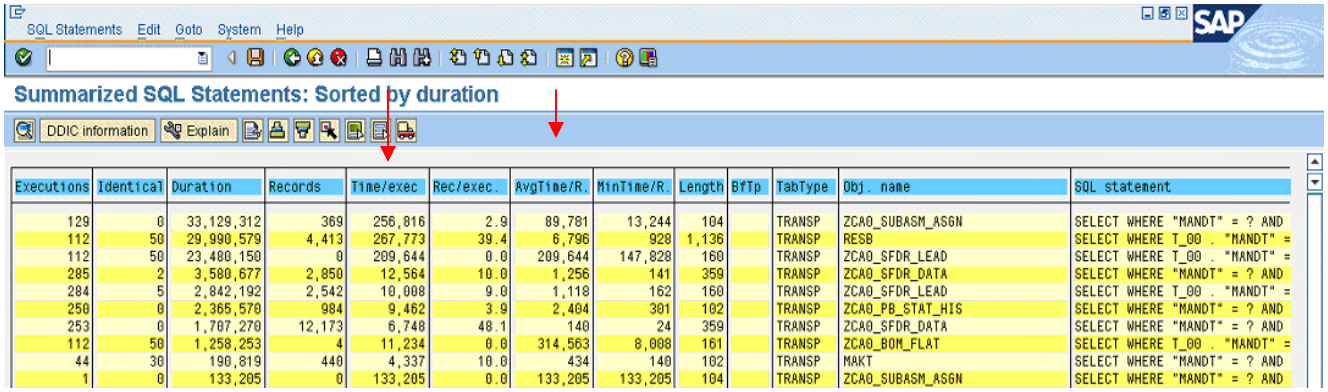


Figure 50: ZCR3 Summarized by SQL Statement

This summary has several useful pieces of information – which statements have (in total) the longest time, whether tables are buffered or not (BfTp), average time per execution and average time per row. With this information, one can distinguish whether a statement takes a lot of time because there are many rows retrieved, or whether retrieving an individual row is slow.

If both “Time/exec” and “AvgTime/R.” (time per row) are slow, the SQL retrieves rows slowly. Here, each call to ZCA0_SUBASM_ASSN averages over 250 ms, and retrieving one row averages 89ms, since there is more than one row on each call to DB2. This is very slow. In contrast, each call to MAKT averages about 4 ms, and retrieving one row averages 434 microseconds, which is ok.

In Figure 50, we see that the largest component of DB request time is selects on the ZCA0_SUBASSM_ASSGN table. If we go back (green arrow) to the screen in Figure 49, and then search for ZCA0_SUBASSM_ASSGN table in the trace, we see repeated slow selects on the table – each call takes several hundred milliseconds to return 5-10 rows.

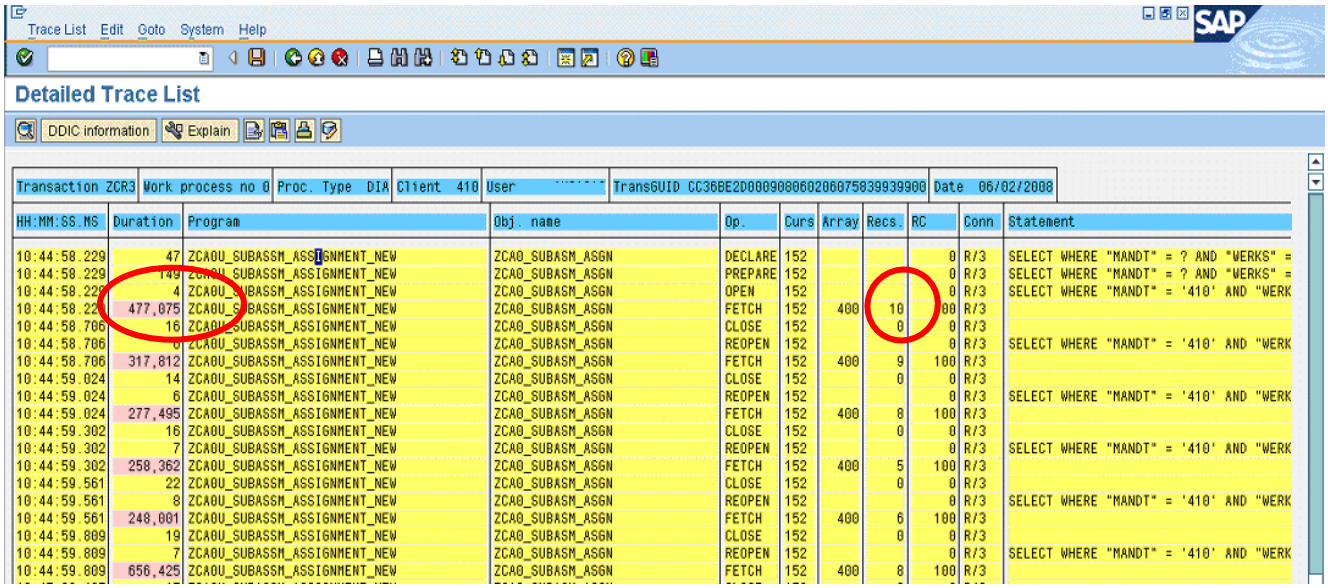


Figure 51: ZCR3 ZCA0_SUBASM_ASSGN

Place the cursor on an OPEN or PREPARE line, and select “Goto > Display ABAP source” to see the ABAP source code. The program name starts with Z, so is custom ABAP.

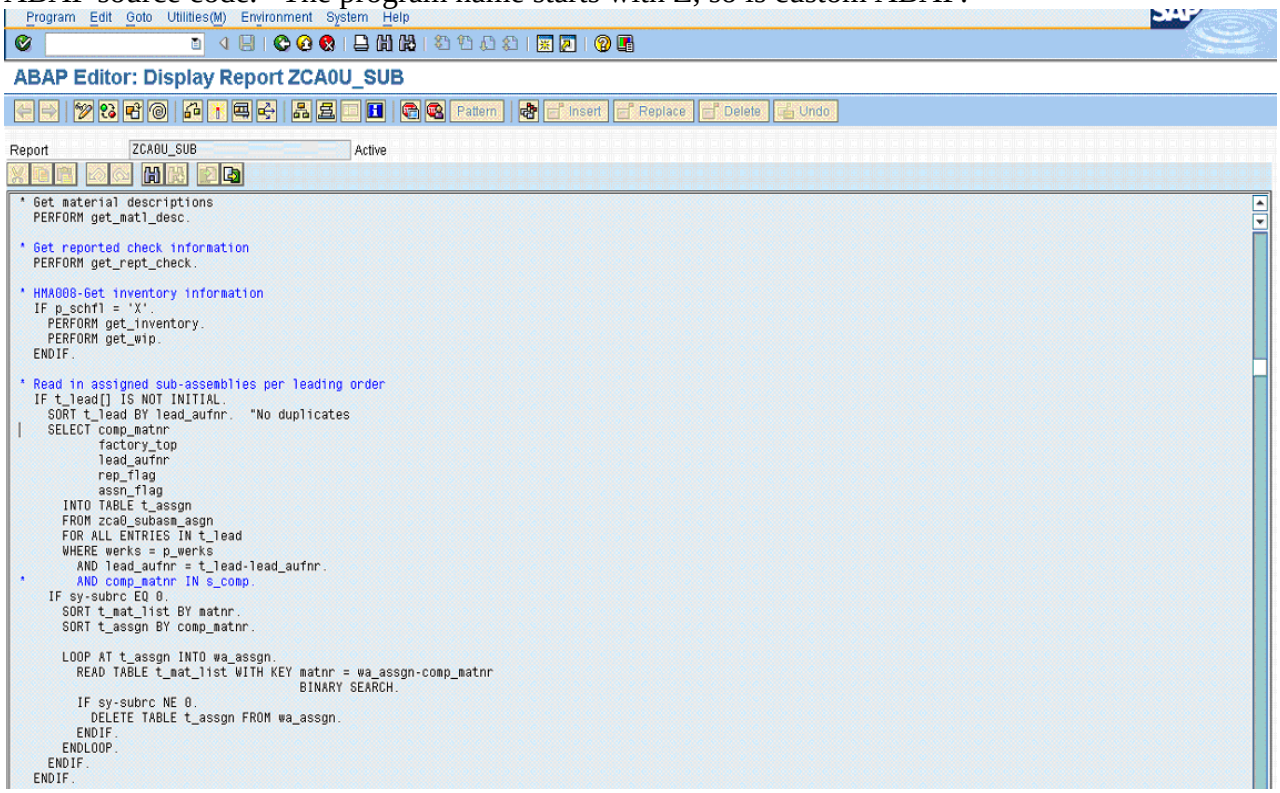


Figure 52: ZCR3 - ABAP source

Again from the SQL trace in Figure 49, to see the statement and values of parameters, place the cursor on an OPEN line, and click the clipboard (Replace placeholder in SQL statement) button.

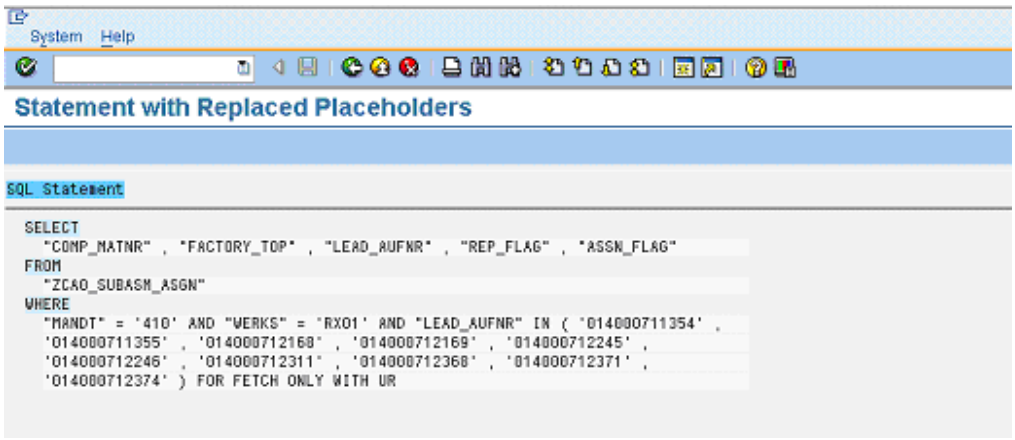


Figure 53: ZCR3 statement with variables

In the SQL statement, there are local predicates (column operation value) on MANDT=, LEAD_AUFNR IN, and WERKS=. To execute this statement most efficiently in the database, one would need an index where the first three columns of the index contain MANDT, WERKS, and LEAD_AUFNR.

To explain the SQL, going back to the trace displayed in Figure 51, place the cursor on one of the OPEN or PREPARE rows, and press “Explain”.

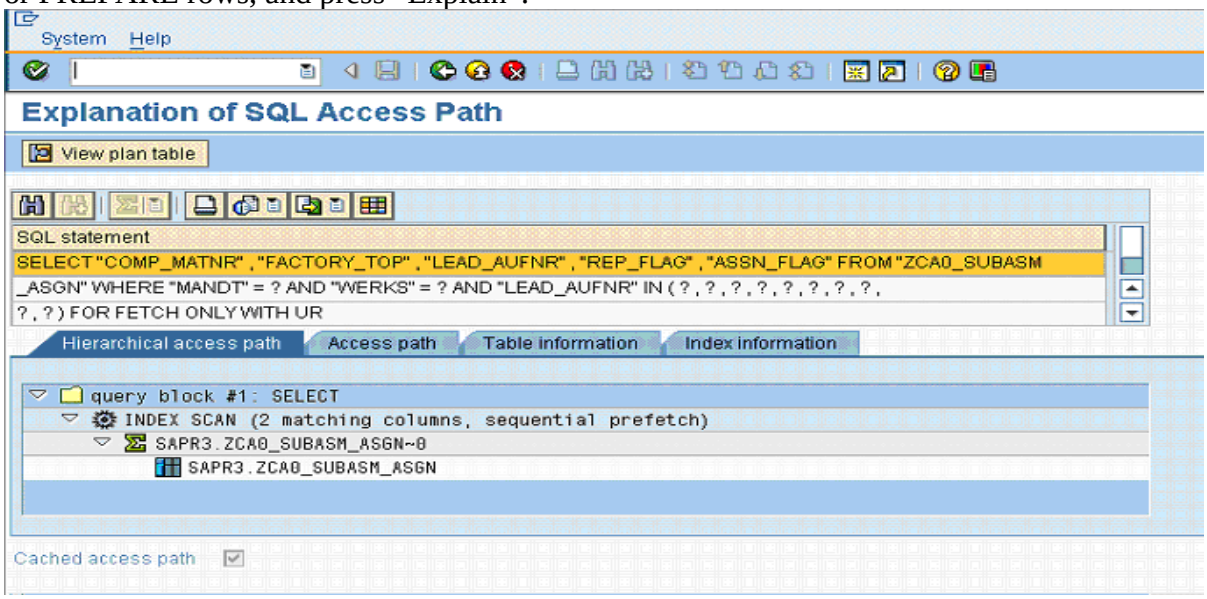


Figure 54: ZCR3 Explain

The explain shows that only two index columns (of the three local predicate columns) are matched. We can display the indexes and index columns with the “Index information” tab.

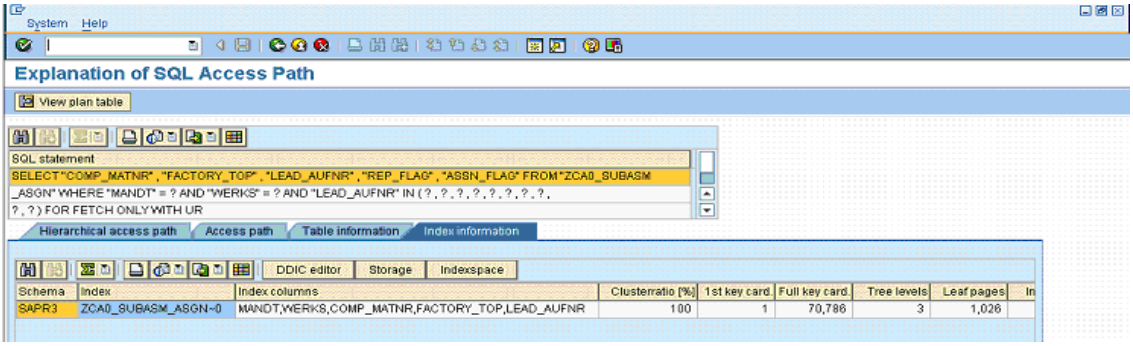


Figure 55: ZCR3 Index display

MANDT and WERKS are the first two index columns, so DB2 can do index matching access on only these two columns. Since there are other index columns between WERKS and LEAD_AUFNR, DB2 does index screening for the LEAD_AUFNR IN local predicate. Index screening is processed in the index, but is not as efficient as matching index access.

We can check the cardinality (number of distinct values) of the columns, to see whether adding a new index (MANDT, WERKS, LEAD_AUFNR) looks like a good choice. (Actually, from the trace in Figure 49, we already know that together the local predicates are filtering, since each call returns just a few rows). Press the "Table information" tab, then select the "ZCA0_SUBASM_ASSGN" table, and press "Col. Card".

There are over 40K distinct values of LEAD_AUFNR in the 70K (inFigure 56) rows of the table.

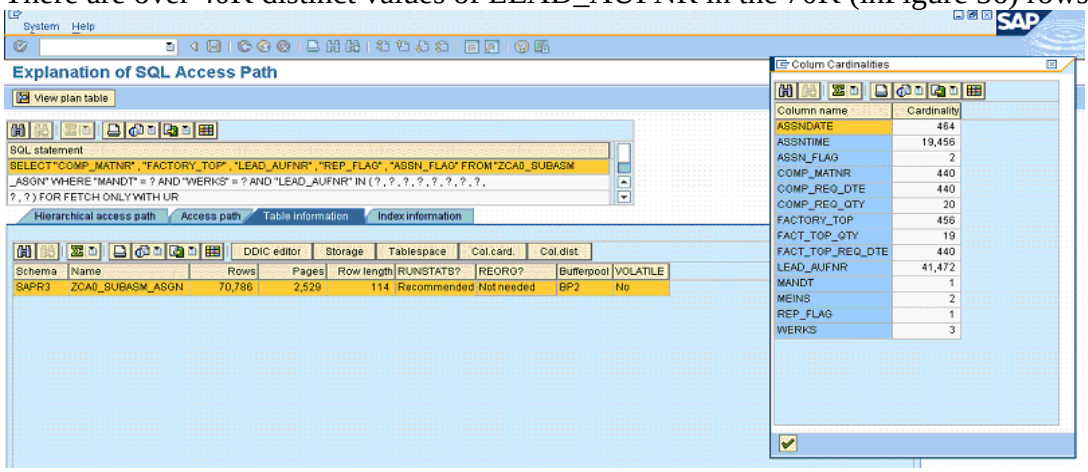


Figure 56: ZCR3 Table information

When the local predicates do not match the indexes, there are several possible ways to approach the problem. See our guidelines in Section 7.5.2.

- Do nothing – if the issue is not critical to the business, and is not causing CPU or I/O problems that impact the system and other users, one may leave it alone. Creating too many custom indexes on a table can impact performance of other programs that update the table

- Modify the ABAP to match the indexes – in this case, if the program already has the values for FACTORY_TOP and COMP_MATNR, so that they could be added to the selection conditions (e.g. factory_top eq xxx) in the ABAP, then DB2 would have local predicates on all the index columns, which would give efficient access. Whenever this is possible, it is a better choice than adding a new index.

- Last, one can add an index - in Figure 56, we see that there are 70K rows in the table, and over 40K distinct values of LEAD_AUFNR. On the average, there will be less than two rows for each value of LEAD_AUFNR, so our select statement with the IN list with ten values will return on average less than 20 rows. With an index containing MANDT, WERKS, and LEAD_AUFNR, one would usually expect to be able to retrieve 20 rows in less than 20 ms (assuming reasonable hit rates), so adding a new index will give a substantial improvement in performance. Based on the summary in Figure 50, the new index would probably reduce DB time by about 25%.

7.5.4. ZCR1 – can the SQL be avoided

This is an example of the type of problem where SQL can be avoided, as is described in Section 7.4.8. We've been asked to investigate the performance of the ZCR1 transaction.

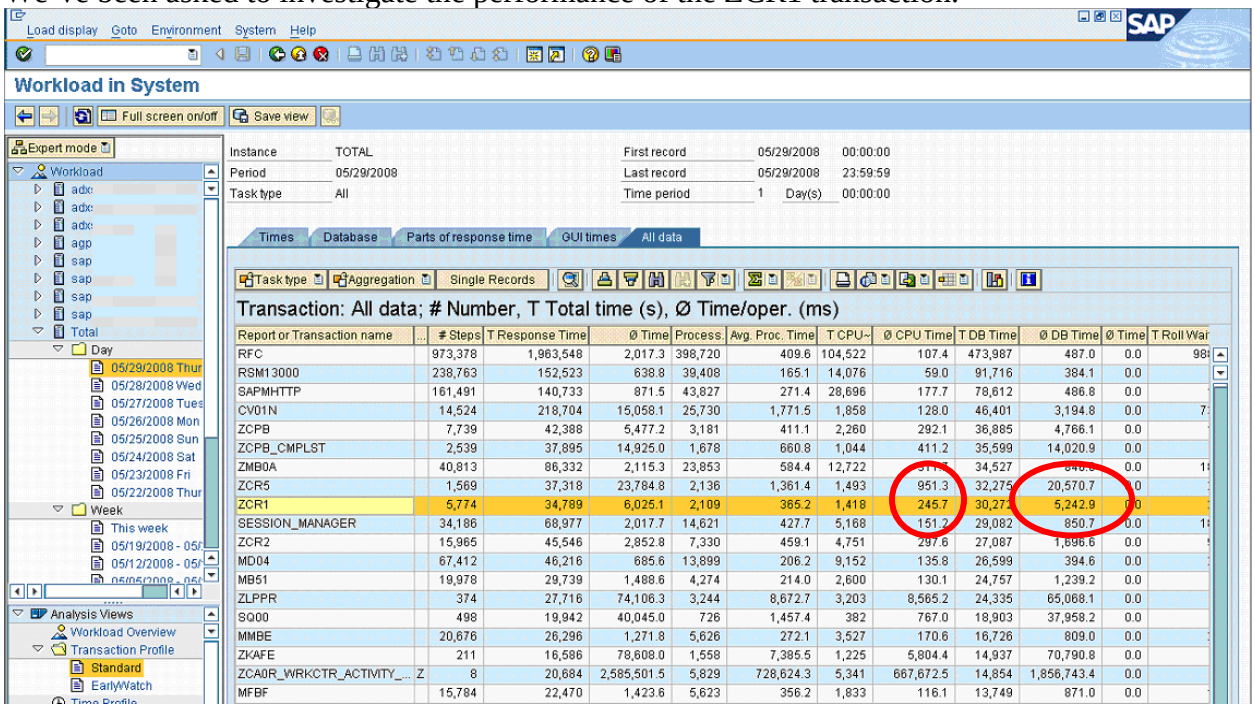


Figure 57: ZCR1 - ST03N statistics

ST03N shows that the transaction is one of the highest impact, in terms of total DB time, and the average DB time (0 DB time – over 5000 ms) is about 20 times as long as average CPU time (0 CPU Time – about 250 ms). Such a large difference between DB time and CPU time is often a sign of a performance problem in retrieving data from the DB server.

Using the STAD transaction, we retrieve some detailed SAP statistics records on the system. We see that some dialog steps run quickly, and that some are slow.

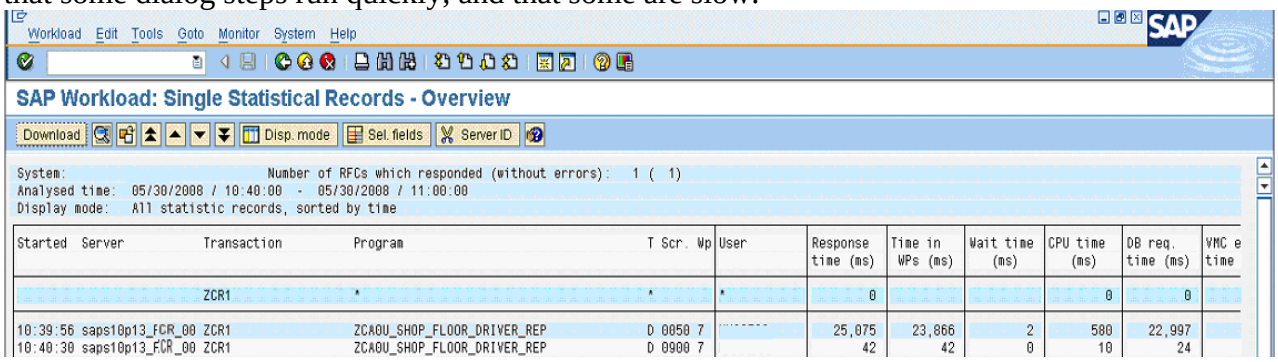


Figure 58: ZCR1 statistics records

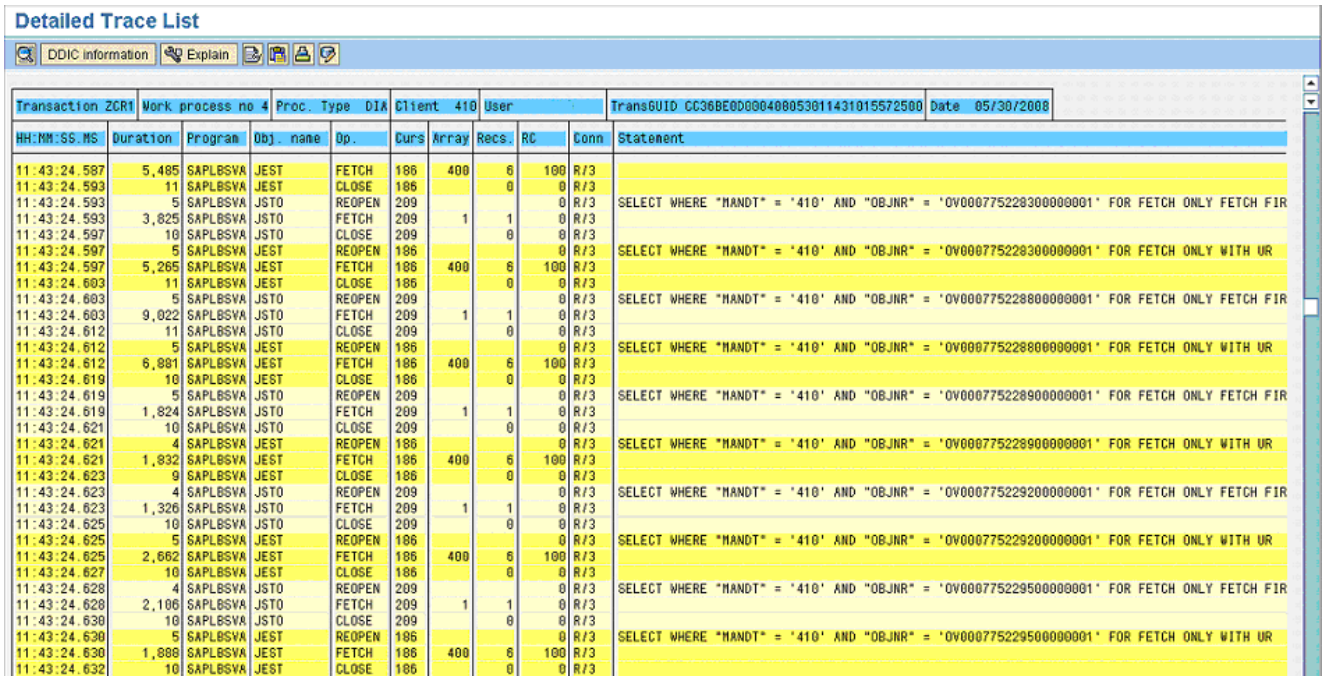
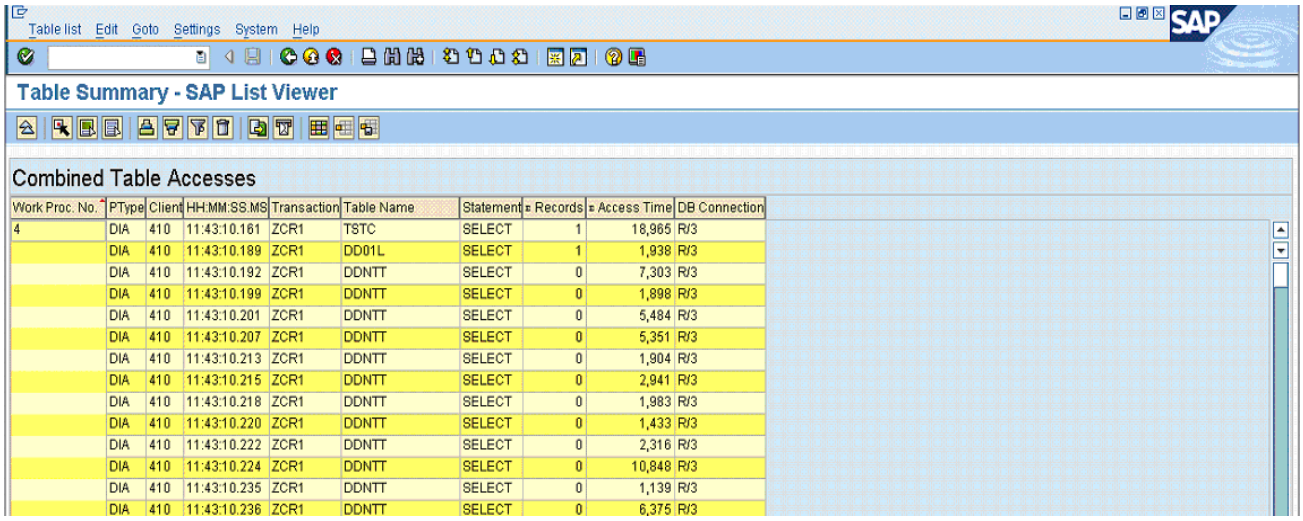


Figure 59: ZCR1 ST05 trace

We trace with ST05, and the initial screen of the formatted report looks fine – average times of a few ms.

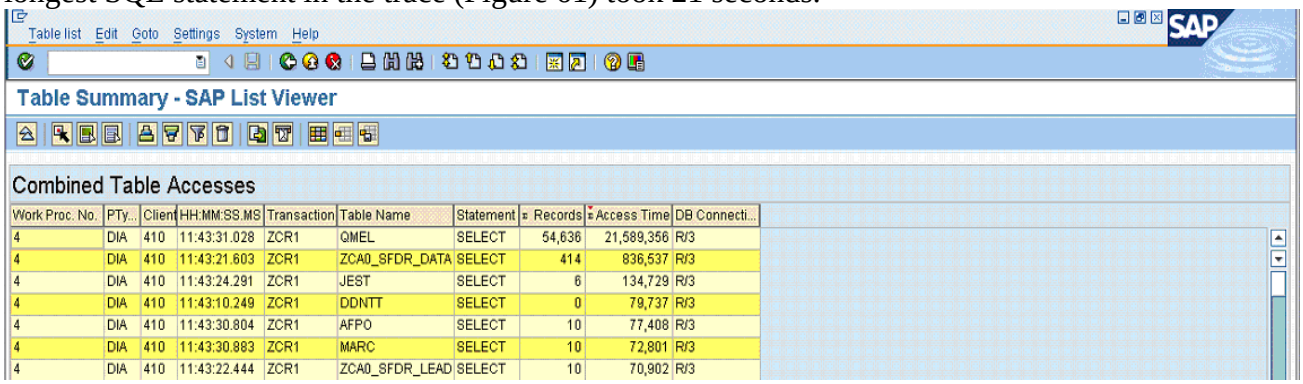
In addition to the “Summarize trace by SQL statement” shown in Figure 50, one can format a trace by selecting “Trace list > Summarize trace by tables”.



Work Proc. No.	PType	Client	HH:MM:SS.MS	Transaction	Table Name	Statement	Records	Access Time	DB Connection
4	DIA	410	11:43:10.161	ZCR1	TSTC	SELECT	1	18,965	R/3
	DIA	410	11:43:10.189	ZCR1	DD01L	SELECT	1	1,938	R/3
	DIA	410	11:43:10.192	ZCR1	DDNTT	SELECT	0	7,303	R/3
	DIA	410	11:43:10.199	ZCR1	DDNTT	SELECT	0	1,898	R/3
	DIA	410	11:43:10.201	ZCR1	DDNTT	SELECT	0	5,484	R/3
	DIA	410	11:43:10.207	ZCR1	DDNTT	SELECT	0	5,351	R/3
	DIA	410	11:43:10.213	ZCR1	DDNTT	SELECT	0	1,904	R/3
	DIA	410	11:43:10.215	ZCR1	DDNTT	SELECT	0	2,941	R/3
	DIA	410	11:43:10.218	ZCR1	DDNTT	SELECT	0	1,983	R/3
	DIA	410	11:43:10.220	ZCR1	DDNTT	SELECT	0	1,433	R/3
	DIA	410	11:43:10.222	ZCR1	DDNTT	SELECT	0	2,316	R/3
	DIA	410	11:43:10.224	ZCR1	DDNTT	SELECT	0	10,848	R/3
	DIA	410	11:43:10.235	ZCR1	DDNTT	SELECT	0	1,139	R/3
	DIA	410	11:43:10.236	ZCR1	DDNTT	SELECT	0	6,375	R/3

Figure 60: ZCR1 - summarized trace

Click on “Access time”, and then press the “Sort in descending order” (inverted pyramid) button. The longest SQL statement in the trace (Figure 61) took 21 seconds.



Work Proc. No.	PTy...	Client	HH:MM:SS.MS	Transaction	Table Name	Statement	Records	Access Time	DB Connecti...
4	DIA	410	11:43:31.028	ZCR1	QMEL	SELECT	54,636	21,589,356	R/3
4	DIA	410	11:43:21.603	ZCR1	ZCA0_SFDR_DATA	SELECT	414	836,537	R/3
4	DIA	410	11:43:24.291	ZCR1	JEST	SELECT	6	134,729	R/3
4	DIA	410	11:43:10.249	ZCR1	DDNTT	SELECT	0	79,737	R/3
4	DIA	410	11:43:30.804	ZCR1	AFPO	SELECT	10	77,408	R/3
4	DIA	410	11:43:30.883	ZCR1	MARC	SELECT	10	72,801	R/3
4	DIA	410	11:43:22.444	ZCR1	ZCA0_SFDR_LEAD	SELECT	10	70,902	R/3

Figure 61: ZCR1 SQL sorted by time

54K rows in 21 seconds is less than ½ ms per row, which is OK. So, we are not going to make the program go faster by speeding up the SQL. We need to look at the ABAP program, so see whether this might be a problem where the program is retrieving more data than it needs.

In the SQL trace, we find the long select on QMEL.

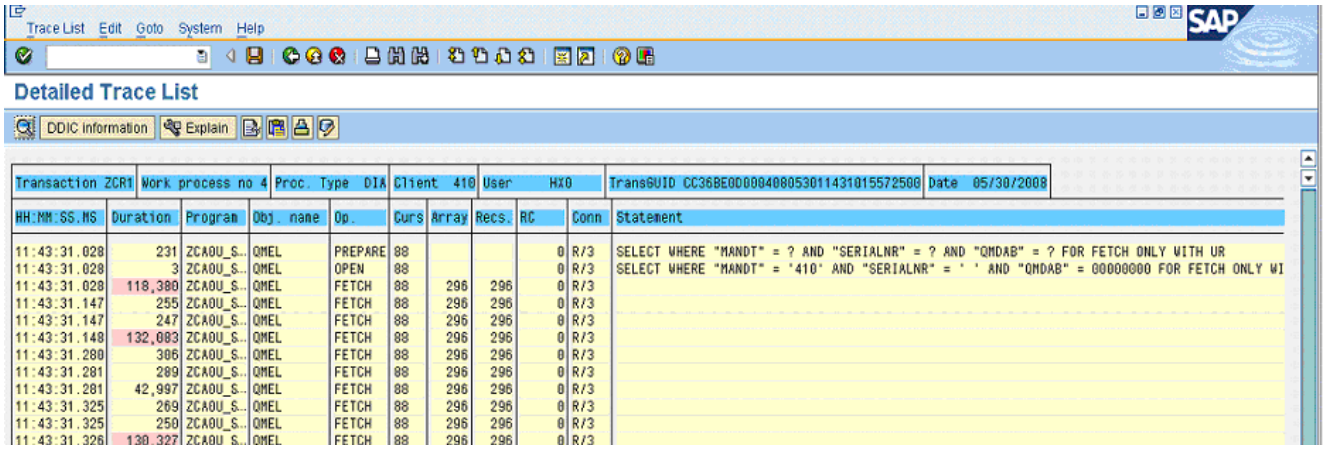


Figure 62: ZCR1 ST05 QMEL

Note that the values being retrieved are MANDT = 410, SERIALNR = ‘ ‘ (space) and QMDAB = 00000000. Perhaps the program needs to retrieve all rows that do not have a serial number assigned.

In the trace in Figure 62, place the cursor on the PREPARE or OPEN line, and select “Goto > Display ABAP source”.

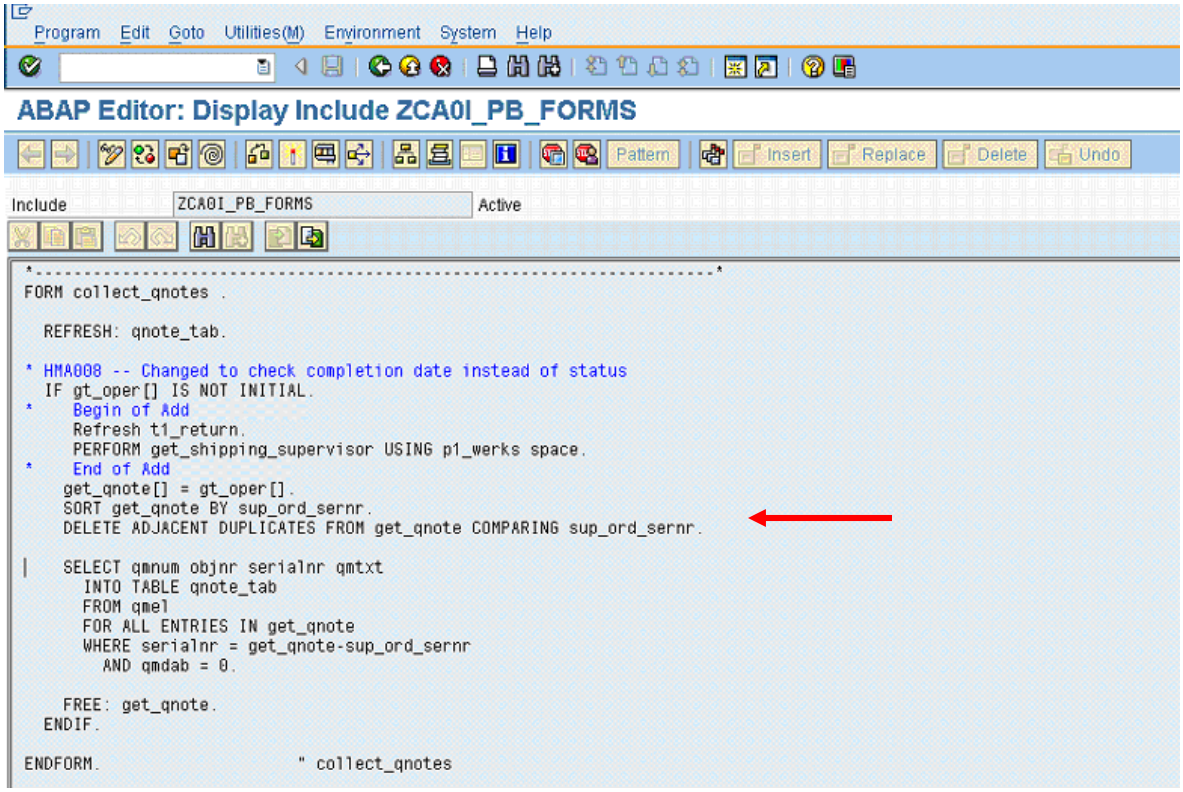


Figure 63: ZCR1 FORM ABAP source

Here, note that the last statement before the SELECT FOR ALL ENTRIES is to delete duplicate entries in the internal table (get_quote) that is used in the FOR ALL ENTRIES. This implies that the internal table should hold values for serialnr, since the program is looking for distinct numbers.

Next, check the source code that calls this FORM in Figure 64. We see the FORM called (PERFORM collect_qnotes), then there is a loop where the quote_tab table is used. But, before quote_tab is read, the program checks that serialnr is NOT INITIAL, that is blank. So, all the rows that were read for SERIALNR = ‘ ‘ are not processed.

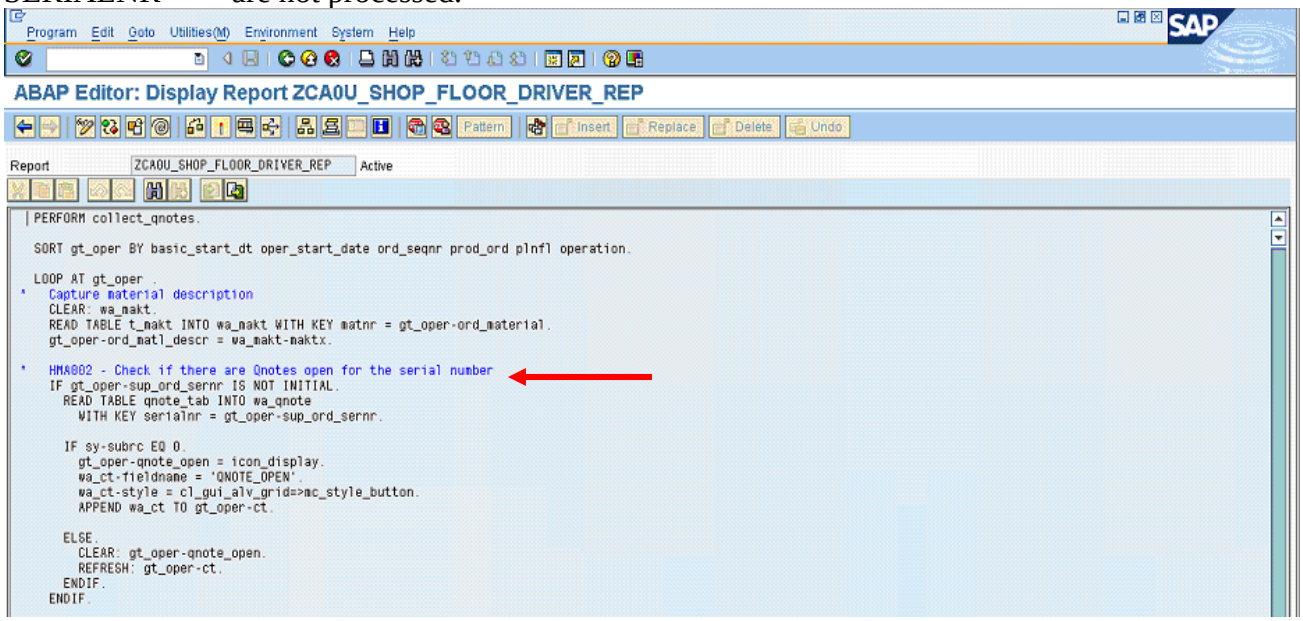


Figure 64: ZCR1 main program ABAP source

At this point, we contact the developers to discuss the program logic, and whether this is a logic error that needs to be fixed, in order to avoid the SQL on QMEL when there are no SERIALNRs to be selected. The internal table is not empty, but contains a value (spaces) that will never match rows that are needed.

7.5.5. ZLTRUCK – can the SQL be avoided

As in Section 7.5.4, this is an example of a situation when the problem is not slow SQL, but SQL that is not needed.

We’re investigating the performance of the transaction ZLTRUCK, where the majority of dialog step elapsed time is DB time. In this STAT record, the response time is 115 seconds, of which 102 seconds is DB request time.

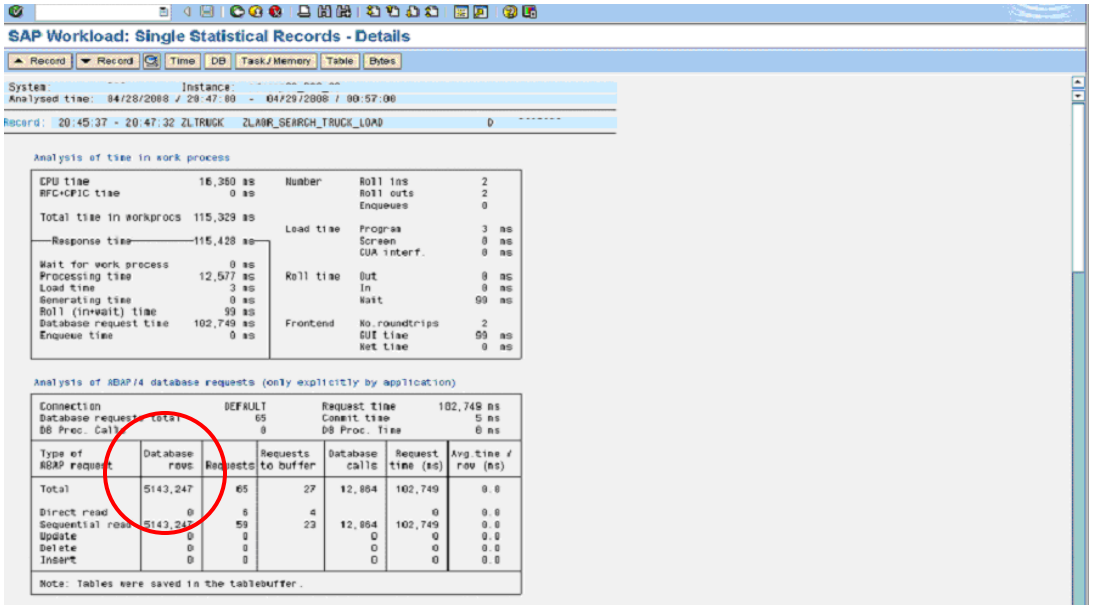


Figure 65: ZLTRUCK statistics record

Here, DB request time is 102 seconds, while CPU time is 16 seconds. But interestingly, there are over 5 million rows read, which is unusual for a dialog transaction.

Stat/tabrec is enabled, press the “Table” button in Figure 65 and we see that LTAP is the table from which 5 million rows are fetched.

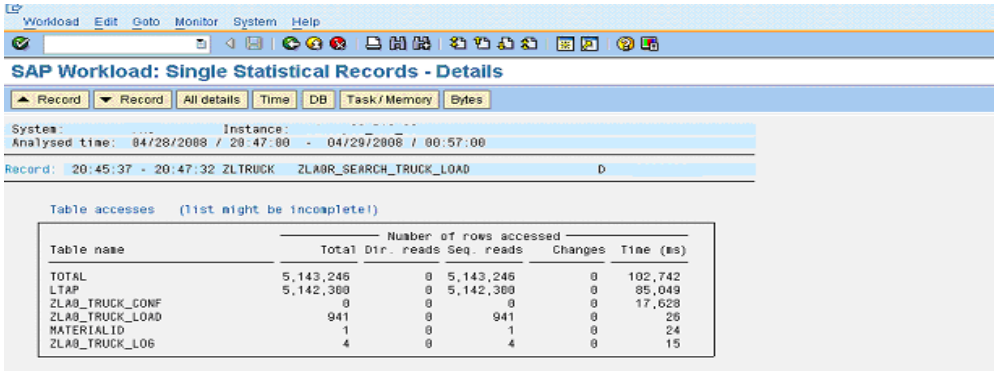


Figure 66: ZLTRUCK stat/tabrec

Now we can run ZLTRUCK, and trace it with ST05, to find the calls to LTAP, and determine why it is selecting so many rows. As in the previous examples, trace and explain the SQL with ST05.

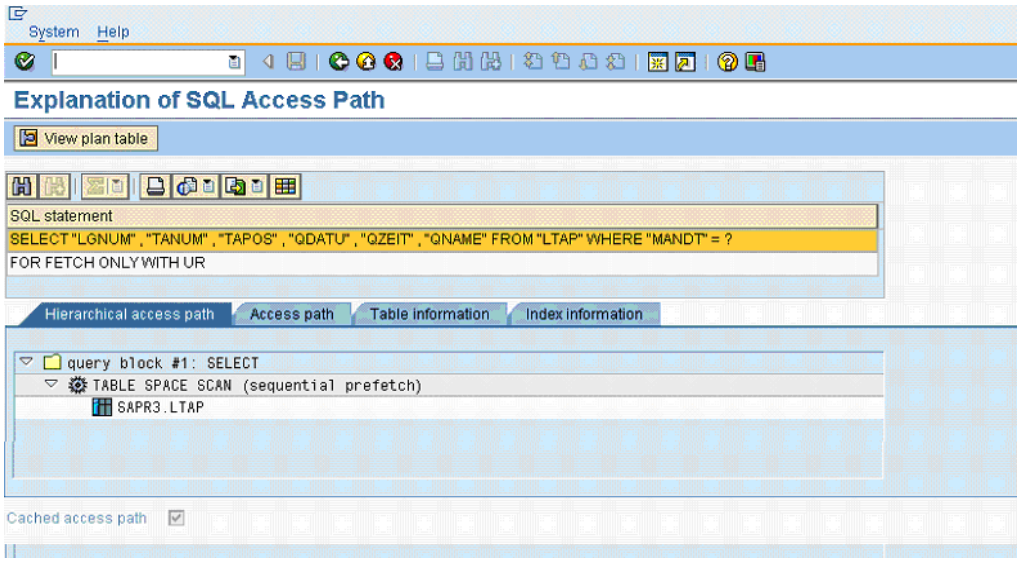


Figure 67: ZLTRUCK LTAP explain

We see that the only local predicate is MANDT=, which is unusual. Normally, a program will specify some subset of the data to be retrieved with where clauses on columns.

When we display the APAB, we see the cause of the problem. The program does a SELECT FOR ALL ENTRIES, which uses an internal table to list the keys of the rows to be retrieved. But, if the internal table is empty, SAP creates the SQL on the table without local predicates on the columns using the internal table for the selection value. Put another way, FOR ALL ENTRIES restricts the list of rows, and with an empty table, there are no restrictions.

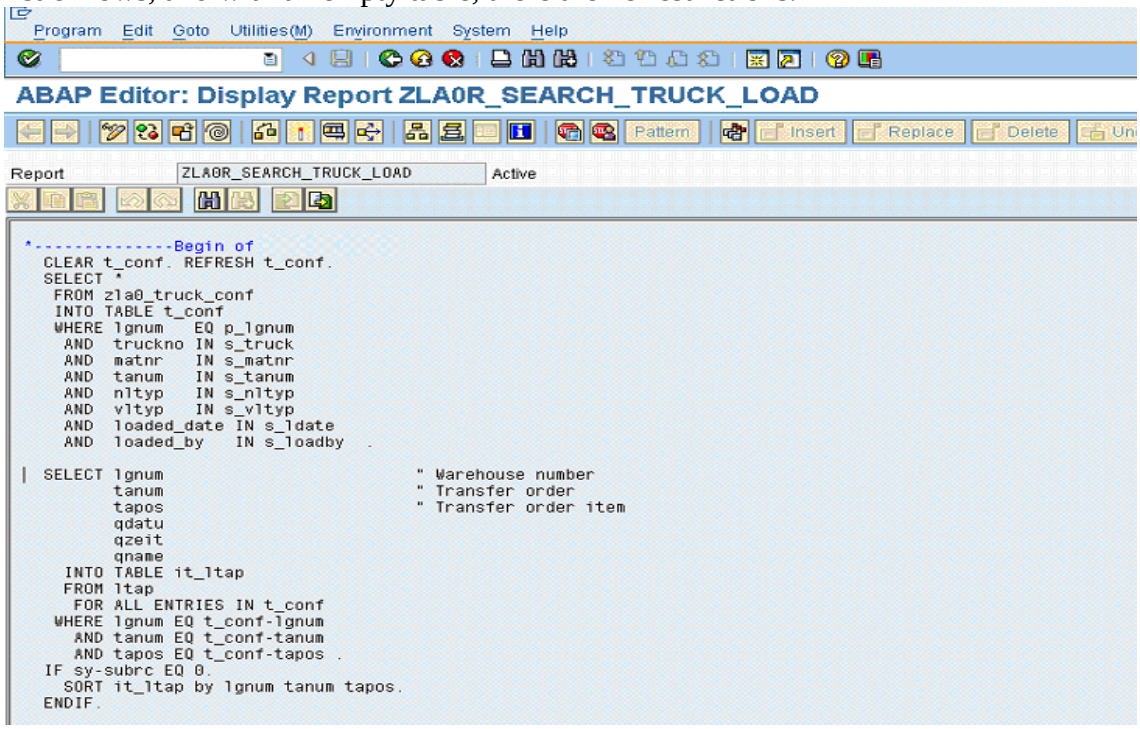


Figure 68: ZLTRUCK ABAP

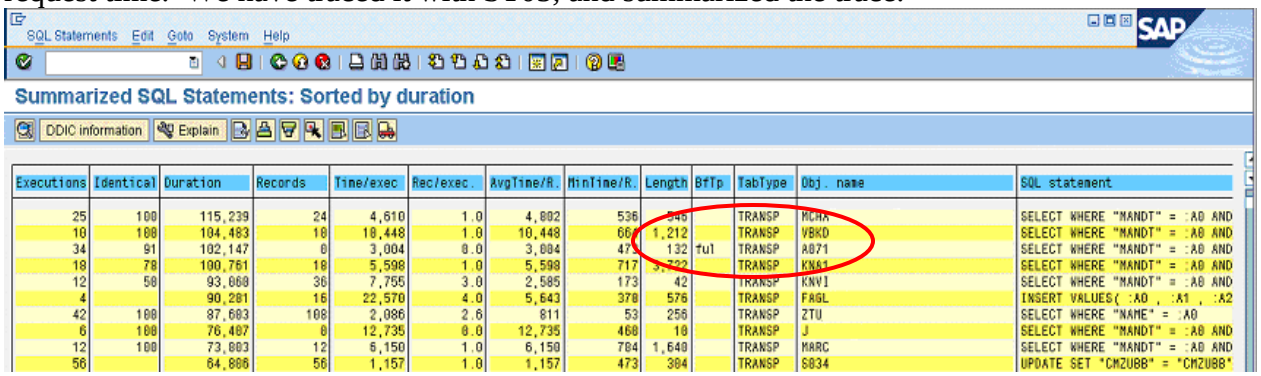
In Figure 68, lgnum, tanum, tapos get their values from the internal table t_conf, so are not included in the SQL if the table is empty. This leaves MANDT=, which is implicity added to the SQL by ABAP.

The result is that the SQL sent to DB2 has one local predicate, MANDT=, even though the program did not specify any rows to retrieve in the internal table.

The cure for this problem is to add a check in the ABAP for whether the internal table is empty or not. If it is not empty, then the SQL can be executed, otherwise, skip the SELECT FOR ALL ENTRIES.

7.5.6. A071 – can the SQL be avoided

We have been asked to review performance of a program which has a large percentage of DB request time. We have traced it with ST05, and summarized the trace.



Executions	Identical	Duration	Records	Time/exec	Rec/exec.	AvgTime/R.	MinTime/R.	Length	BfTp	TabType	Obj. name	SQL statement
25	100	115,239	24	4,610	1.0	4,802	536	345		TRANSP	MCHA	SELECT WHERE "MANDT" = :A0 AND
10	100	104,483	10	10,448	1.0	10,448	66	1,212		TRANSP	VBKD	SELECT WHERE "MANDT" = :A0 AND
34	91	102,147	0	3,004	0.0	3,004	47	132	ful	TRANSP	A071	SELECT WHERE "MANDT" = :A0 AND
18	78	100,761	18	5,598	1.0	5,598	717	3,722		TRANSP	KNA1	SELECT WHERE "MANDT" = :A0 AND
12	58	93,060	36	7,755	3.0	2,585	173	42		TRANSP	KNVI	SELECT WHERE "MANDT" = :A0 AND
4		90,281	16	22,570	4.0	5,643	378	576		TRANSP	FAGL	INSERT VALUES(:A0 , :A1 , :A2
42	100	87,603	100	2,086	2.6	811	53	256		TRANSP	ZTU	SELECT WHERE "NAME" = :A0
6	100	76,407	0	12,735	0.0	12,735	460	10		TRANSP	J	SELECT WHERE "MANDT" = :A0 AND
12	100	73,803	12	6,150	1.0	6,150	704	1,640		TRANSP	MARC	SELECT WHERE "MANDT" = :A0 AND
56		64,806	56	1,157	1.0	1,157	473	304		TRANSP	S034	UPDATE SET "CMZUBB" = "CMZUBB"

Figure 69: A071 summarized SQL trace

In Figure 69, note that the third entry (A071) is set to be buffered (BfTp = Ful), but the program is making calls to the DB server to retrieve rows. Normally, buffered tables should be a very small percent of DB time, since the data is usually already in the SAP buffers.

DB calls to buffered tables can happen if the generic buffers on the application server are too small, such that the table does not fit in memory on the application server. We run ST02 to display the buffer activity.

Buffer	HitRatio %	Alloc. KB	Freesp. KB	% Free Sp.	Dir. Size	FreeDirEnt	% Free Dir	Swaps	DB Accs
Nametab (NTAB)								0	
Table definition	99.97	13,260	1,003	9.02	39,000	3,517	9.02	0	716
Field definition	99.95	99,047	5,192	5.41	39,000	22,698	58.20	31	893
Short NTAB	100.00	4,219	1,166	38.87	9,750	3,872	39.71	0	5,878
Initial records	88.58	11,719	9,207	87.69	9,750	1,578	16.18	389	9,599
								0	
program	99.89	800,000	6,010	0.79	200,000	178,064	89.03	66	335
CUA	98.54	10,000	775	9.45	5,000	4,509	90.18	47	135
Screen	99.56	19,532	4,461	24.09	10,000	9,457	94.57	21	104
Calendar	100.00	488	445	98.23	480	385	96.25	0	15
OTR	99.95	4,096	3,367	99.79	2,000	1,998	99.90	0	
								0	
Tables								0	
Generic Key	99.43	126,954	6,417	5.77	50,000	8,761	17.52	36,147	28,450,321
Single record	99.89	60,000	3,400	5.70	1,000	726	72.60	15	540
								0	
Export/Import	87.01	40,000	14,499	44.19	20,000	14,012	70.06	942	
Exp./ Imp. SHM	99.41	4,096	3,226	95.61	2,000	1,996	99.80	0	

Figure 70: ST02

In ST02 shown in Figure 70, note that there are many DB calls for data that is in the generic buffer.

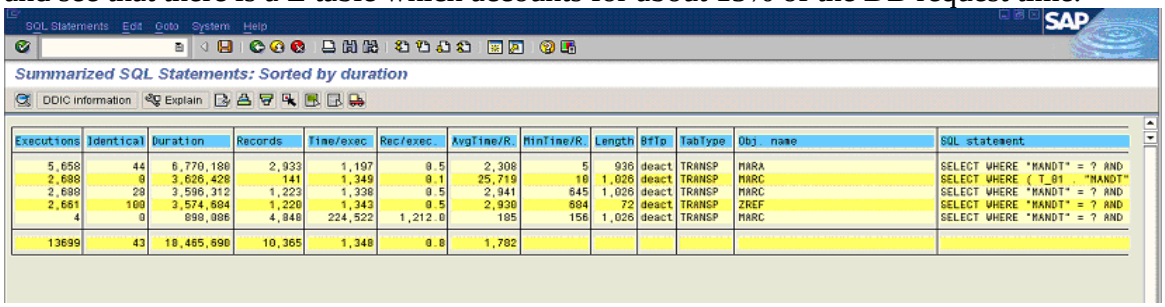
See section 9.2.4 for information on using ST02 to check if the generic buffer is too small.

If the generic buffer area is too small, then tables that should be buffered cannot fit in buffer memory on the application server, and this will cause performance problems, and excessive DB calls.

There can also be DB calls for buffered tables if the ABAP uses “BYPASSING BUFFER” option. Display the ABAP from the ST05 trace, to check if BYPASSING BUFFER is specified.

7.5.7. ZREF – can the SQL be avoided

We’ve been asked to investigate the performance of a program. We trace and summarize the SQL, and see that there is a Z table which accounts for about 15% of the DB request time.



Executions	Identical	Duration	Records	Time/exec.	Rec/exec.	AvgTime/R.	MinTime/R.	Length	BfTp	TabType	Obj. name	SQL statement
5,658	44	6,770,188	2,933	1,197	0.5	2,308	5	936	deact	TRANSP	MARA	SELECT WHERE "MANDT" = ? AND
2,698	0	3,626,428	141	1,349	0.1	25,719	10	1,026	deact	TRANSP	MARC	SELECT WHERE (T_01 . "MANDT"
2,698	28	3,596,312	1,223	1,338	0.5	2,941	645	1,026	deact	TRANSP	MARC	SELECT WHERE "MANDT" = ? AND
2,651	100	3,574,694	1,220	1,343	0.5	2,930	694	72	deact	TRANSP	ZREF	SELECT WHERE "MANDT" = ? AND
4	0	898,986	4,848	224,522	1,212.0	185	156	1,026	deact	TRANSP	MARC	SELECT WHERE "MANDT" = ? AND
13699	43	18,465,690	10,365	1,348	0.8	1,782						

Figure 71: ZREF ST05 summary

In Figure 71, note that the table ZREF accounts for a bit more than 15% of the DB request time, and it is not set to be buffered. Since it is a custom table, it may be that the buffering was not set correctly.

Check the table accesses with ST10, to see whether it might also be a system-wide candidate for buffering. For more information on buffering tables in SAP, see Section 9.2.5.

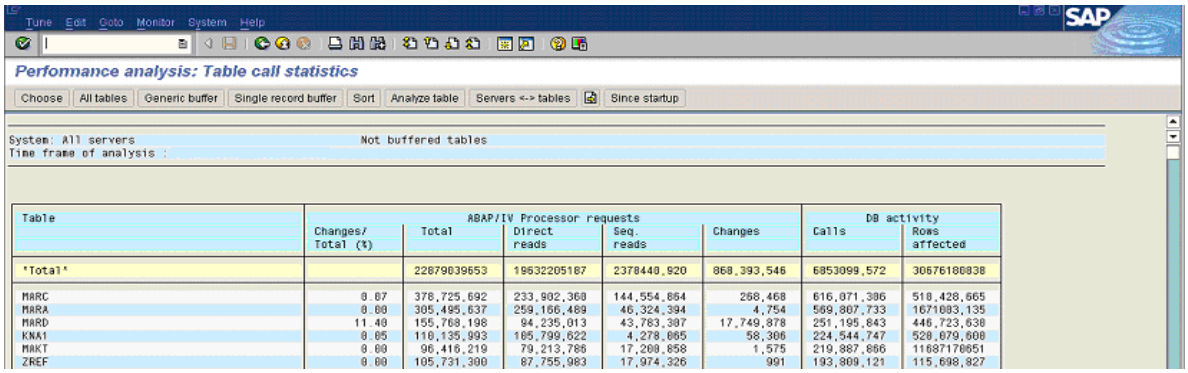


Figure 72: ZREF ST10

The majority of selects are via select single (direct read) but about 20% are select. Generic buffering would support both select single and select, single record buffer would only support the select single.

Use DB05, to check the size of the table and its key columns for generic buffering.

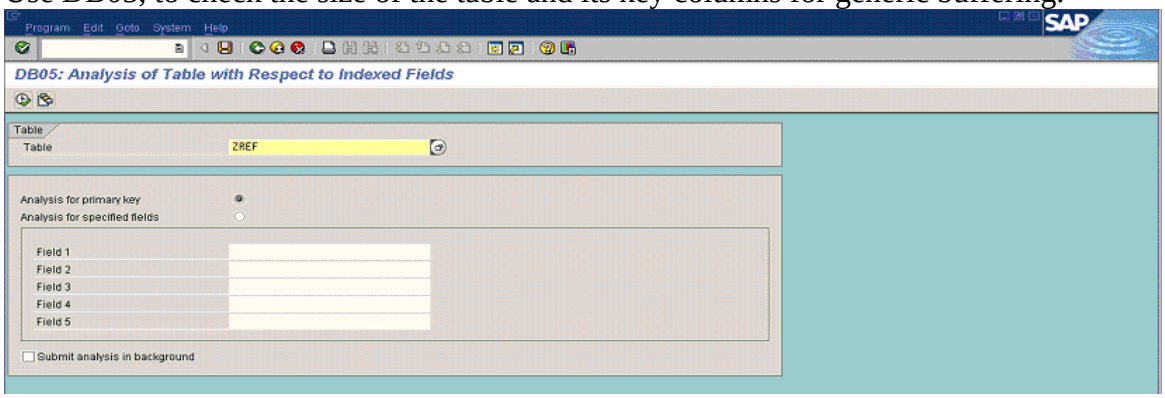


Figure 73: ZREF DB05

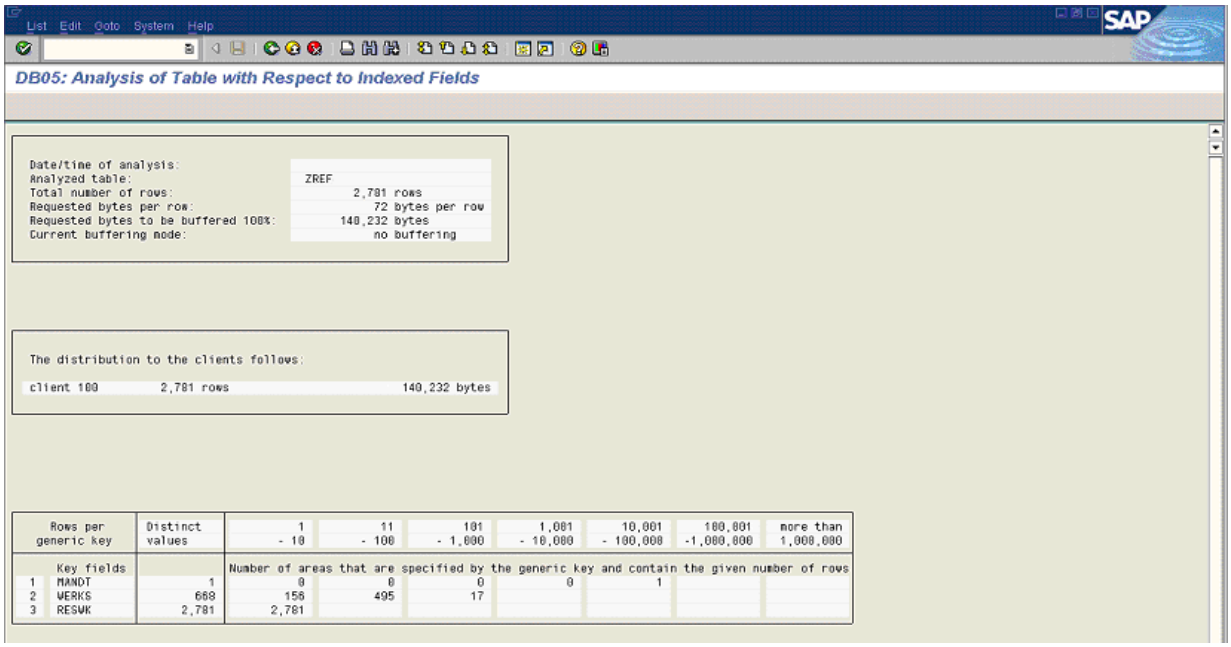


Figure 74: ZREF DB05 data

The table has few rows, and would only require 140K for full buffering. It is not currently buffered, but it looks like a good candidate for buffering – it is relatively small and seldom changed. One could buffer it generic on two columns (mandt, werks), or fully buffer the table to offload the DB requests to the application server.

One would need to discuss with the application developer whether this table meets the third criterion for buffering – can the application tolerate a short interval where the data is out of sync on the app servers. When buffered tables are changed, it takes minute or so (depending on the SAP parameter settings) to propagate the change to all app servers. For customization data, it is usually OK to have this small interval when the data is not the same.

For business data like stock on hand of a material or an account balance, one cannot have different information on different app servers, and these types of data would not be buffered in SAP.

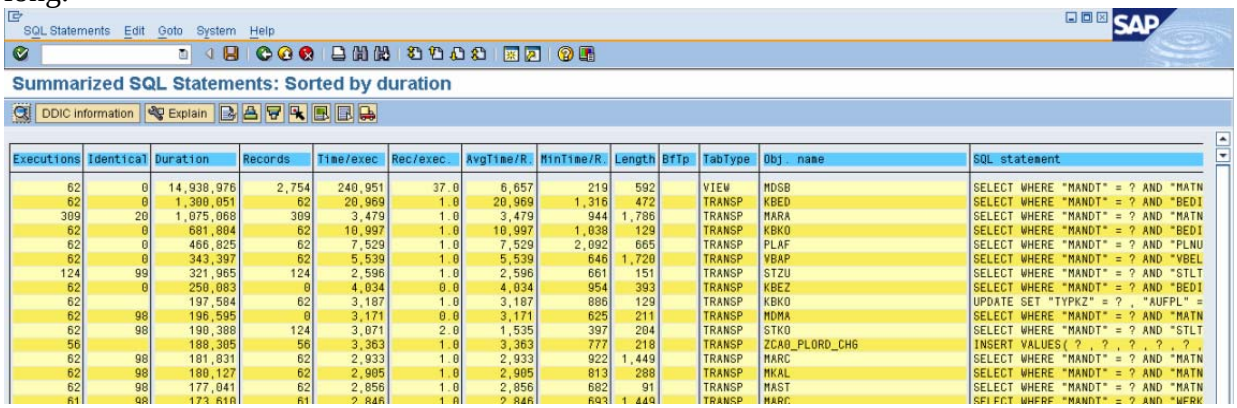
7.5.8. Changing clustering sequence to optimize access times

In this example, the index supports all the local predicates, but SQL is still slow.

DB2 can cluster a table, which means that it places the rows in the table in order based on the values in a column or columns. SAP delivers the system with clustering defined on the ~0 index of each table, but in order to optimize a specific business process, the clustering sequence may need to be changed.

Please note that when the clustering sequence is changed, programs that have SQL that uses the former clustering index will probably run slower. Clustering should be changed only after evaluating the performance constraints of a business process, and considering the impact on programs that will lose the benefit of the original clustering sequence.

In this example, we have traced a program and found that the largest part of DB delay is selects on the MDSB view (RESB table), where each row takes about 6 ms to retrieve. 6 ms per row is rather long.



Executions	Identical	Duration	Records	Time/exec	Rec/exec.	AvgTime/R.	MinTime/R.	Length	BfTp	TabType	Obj. name	SQL statement
62	0	14,938,976	2,754	240,951	37.0	6,657	219	592		VIEW	MDSB	SELECT WHERE "MANDT" = ? AND "MATN
62	0	1,300,051	62	20,969	1.0	20,969	1,316	472		TRANSP	KBED	SELECT WHERE "MANDT" = ? AND "BEDI
309	20	1,075,068	309	3,479	1.0	3,479	944	1,786		TRANSP	MARA	SELECT WHERE "MANDT" = ? AND "MATN
62	0	681,804	62	10,997	1.0	10,997	1,038	129		TRANSP	KBKO	SELECT WHERE "MANDT" = ? AND "BEDI
62	0	466,825	62	7,529	1.0	7,529	2,092	665		TRANSP	PLAF	SELECT WHERE "MANDT" = ? AND "PLNU
62	0	343,397	62	5,539	1.0	5,539	646	1,720		TRANSP	VBAP	SELECT WHERE "MANDT" = ? AND "VBEL
124	99	321,965	124	2,596	1.0	2,596	661	151		TRANSP	STZU	SELECT WHERE "MANDT" = ? AND "STLT
62	0	250,083	0	4,034	0.0	4,034	954	393		TRANSP	KBEZ	SELECT WHERE "MANDT" = ? AND "BEDI
62		197,584	62	3,187	1.0	3,187	886	129		TRANSP	KBKO	UPDATE SET "TYPKZ" = ? , "AUFPL" =
62	98	196,595	0	3,171	0.0	3,171	625	211		TRANSP	MDMA	SELECT WHERE "MANDT" = ? AND "MATN
62	98	190,388	124	3,071	2.0	1,535	397	204		TRANSP	STKO	SELECT WHERE "MANDT" = ? AND "STLT
56		188,305	56	3,363	1.0	3,363	777	218		TRANSP	ZCA0_PLORD_CH6	INSERT VALUES(? , ? , ? , ? , ? ,
62	98	181,831	62	2,933	1.0	2,933	922	1,449		TRANSP	MARC	SELECT WHERE "MANDT" = ? AND "MATN
62	98	180,127	62	2,905	1.0	2,905	813	288		TRANSP	MKAL	SELECT WHERE "MANDT" = ? AND "MATN
62	98	177,041	62	2,856	1.0	2,856	682	91		TRANSP	MAST	SELECT WHERE "MANDT" = ? AND "MATN
61	98	173,618	61	2,846	1.0	2,846	693	1,449		TRANSP	MARC	SELECT WHERE "MANDT" = ? AND "WERK

Figure 75: Slow selects on MDSB

When we explain the statement, we see that the RESB~M index is being used to access the data, with five columns in the index.

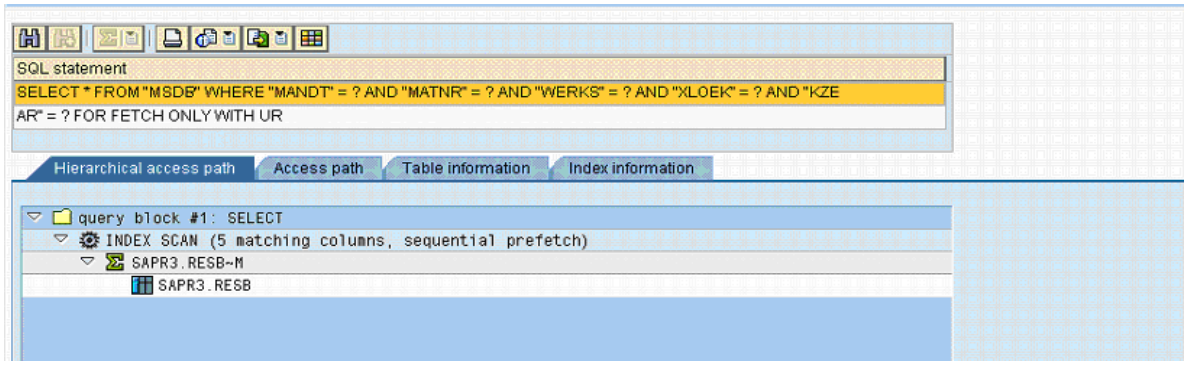
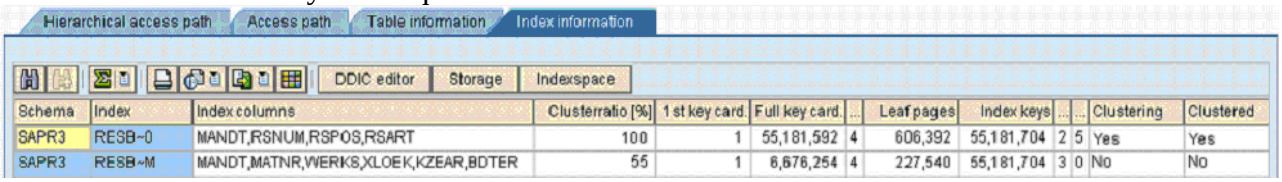


Figure 76: RESB~M explain

And checking the index statistics, we see that the cluster ratio of the RESB~M index is low, which means that it is not likely that sequential rows in the index are near to each other in the table.



Schema	Index	Index columns	Clusterratio [%]	1 st key card.	Full key card.	...	Leaf pages	Index keys	...	...	Clustering	Clustered
SAPR3	RESB~0	MANDT,RSNUM,RSPOS,RSART	100	1	55,181,592	4	606,392	55,181,704	2	5	Yes	Yes
SAPR3	RESB~M	MANDT,MATNR,WERKS,XLOEK,KZEAR,BDTER	55	1	6,676,254	4	227,540	55,181,704	3	0	No	No

Figure 77: RESB clustering

With the data we have so far, we know the statement is slow (6 ms per row), but we cannot see what the impact is of I/O delay on the statement. The ST05 trace shows statement elapsed time, but not the components (CPU, I/O, locks, network, etc) of the elapsed time. To see the time components, we need to use ST04 and find the statement in cache.

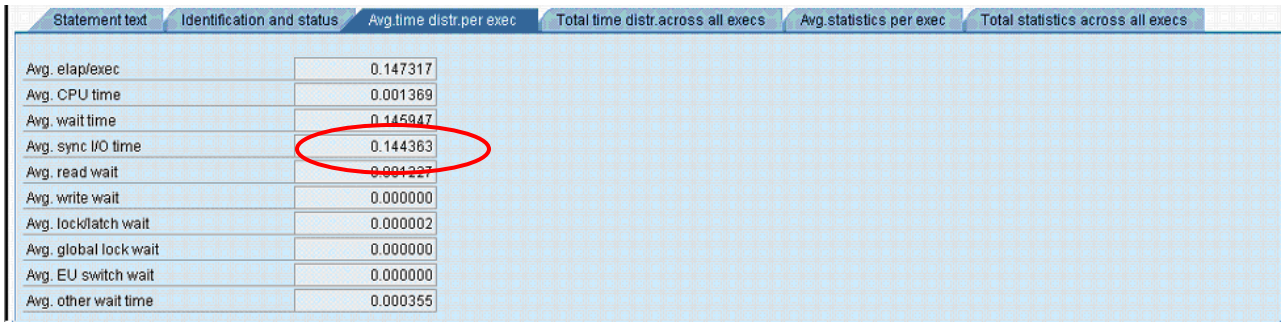


Figure 78: RESB time per execution

Each execution averages about 150 ms, and almost all is synchronous I/O delay.

When you compare an ST05 trace and ST04 statement statistics, the elapsed times will be different, since ST04 is an average of all executions since the statement was prepared, and ST05 is a subset of DB calls.

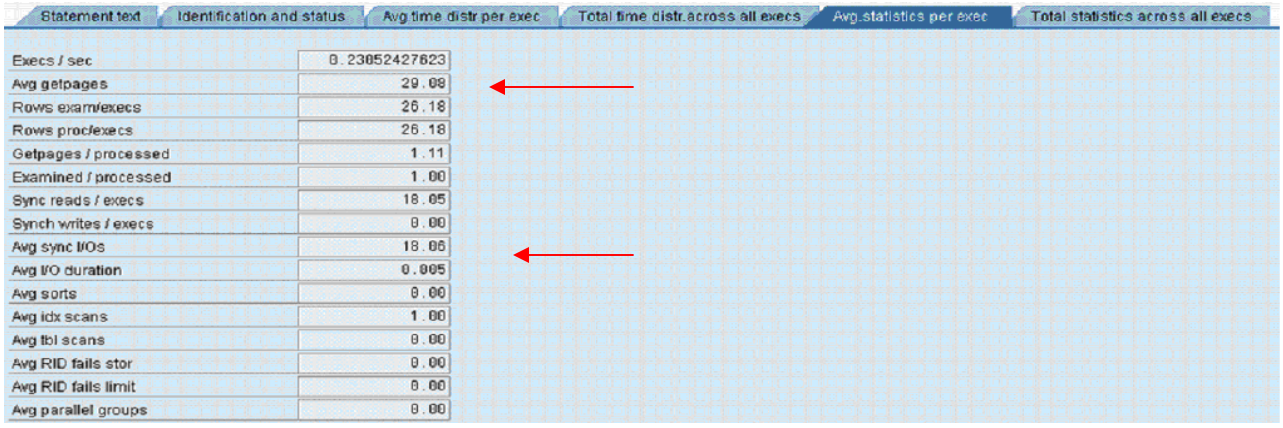


Figure 79: RESB statistics per exec

Note that rows examined and rows processed are the same, which is good. This means that none of the local predicates are applied on the table, they are all applied in the index. But, there are 18 synch I/Os for each execution, and the statement has a hit rate of (26.18-18.05)/26.18 = 31%.

This is a secondary index with low cluster ratio, so rows that are sequential in the index are not sequential in the table. This wide distribution of data causes I/O on the table when rows are read for the result.

Synchronous I/O can also be caused by reading rows with indirect references, which is when rows have been updated and the update could not be saved in the original location. SAP's use of DB2 RTS detects when REORG is needed due to indirect references. The DB2 NEARINDREF and FARINDREF display the number of near and far indirect references. Check whether REORG is needed, before evaluating changes in clustering sequence.

In order to speed up the statement, we can change clustering sequence so that all the rows are clustered by the RESB~M index, and there will be fewer getpages, as well as fewer I/Os per execution.

Here is an example of statement times for a select using RESB~M, when RESB is clustered on this index. Note that the statement hit rate is much higher than in Figure 79.

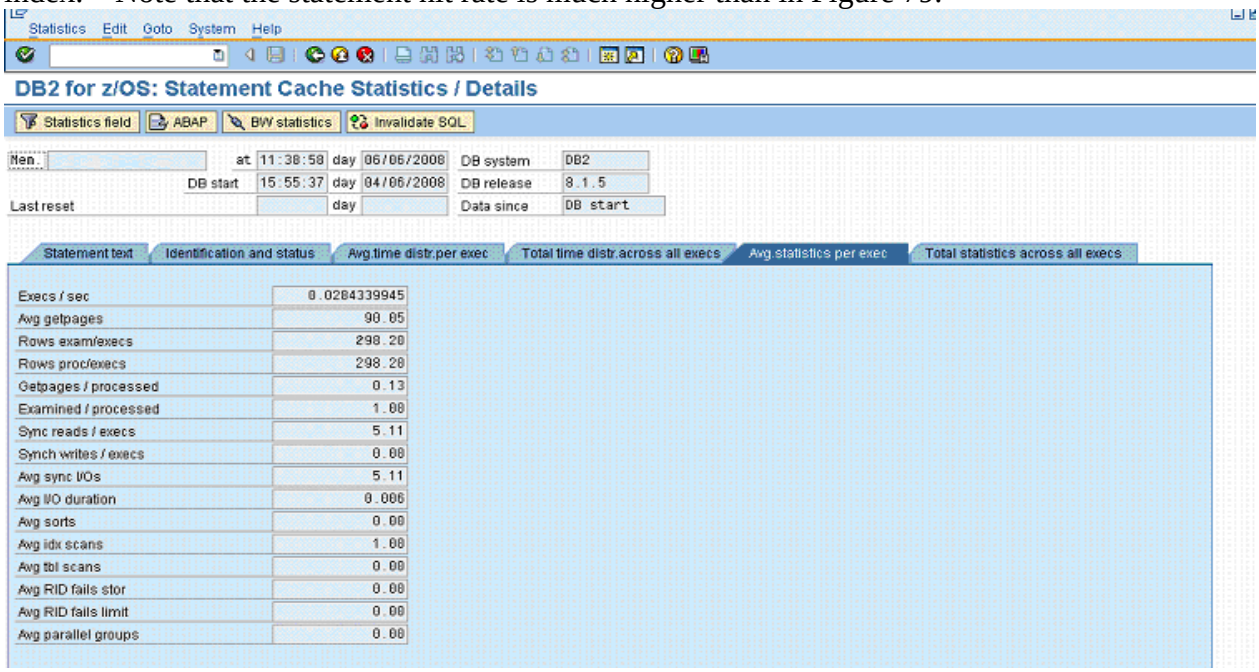


Figure 80: access by RESB~M after clustering on ~M

Now the statement hit rate is over 90% (98 getpages – 5 synch reads) / 98 getpages. So, with the rows clustered on the ~M index, there are fewer I/O operations per execution, and the DB calls will retrieve the rows more quickly.

7.5.9. Modify Program so that data retrieval matches clustering sequence

We've been asked to review performance of a reporting program. The STAT record times are in Figure 81.

Analysis of time in work process

CPU time	122,800 ms	Number	Roll ins	2	
RFC+CPIC time	0 ms		Roll outs	2	
			Enqueues	0	
Total time in workprocs	402,178 ms				
		Load time	Program	4	ms
Response time	402,196 ms		Screen	0	ms
			CUA interf.	0	ms
Wait for work process	1 ms				
Processing time	122,880 ms	Roll time	Out	0	ms
Load time	4 ms		In	0	ms
Generating time	0 ms		Wait	17	ms
Roll (in+wait) time	17 ms				
Database request time	280,294 ms	Frontend	No.roundtrips	2	
Enqueue time	0 ms		GUI time	17	ms
			Net time	0	ms

Figure 81: Financial report STAT times

The majority of time is DB time. The STAT record DB statistics show that each row takes on average 2.7 ms, and if using the data in Figure 82 we divide DB calls (107,190) into DB time (280,294), each call takes about 2.7 ms. So, we do not have a problem with individual calls being extremely long. (Note that we need to calculate call time, since the STAT records show time per row, which is not the same as time per call.)

Analysis of ABAP/4 database requests (only explicitly by application)

Connection	DEFAULT	Request time	280,294 ms
Database requests total	123,395	Commit time	2 ms
DB Proc. Calls	0	DB Proc. Time	0 ms

Type of ABAP request	Database rows	Requests	Requests to buffer	Database calls	Request time (ms)	Avg.time / row (ms)
Total	107,198	123395	63	107,190	280,294	2.6
Direct read	8,057	16146	25		14,831	1.8
Sequential read	99,141	107249	38	107,190	265,463	2.7
Update	0	0		0	0	0.0
Delete	0	0		0	0	0.0
Insert	0	0		0	0	0.0

Figure 82: Financial report STAT DB times

Since the majority of time is DB time, we use ST05 to trace the program when it is running. Note the progression of BELNR values on the calls (they jump up and down). We will determine the table clustering sequence later, and compare the data access pattern to that.

Duration	Obj. name	Op.	Recs.	RC	Statement
5	RFBLG	REOPEN		0	SELECT WHERE "MANDT" = '300' AND "BUKRS" = '3010' AND "BELNR" = '2009969200' AND "GJAHR" = 2009 ORDER
2,780	RFBLG	FETCH	1	100	
8	RFBLG	CLOSE	0	0	
3	RFBLG	REOPEN		0	SELECT WHERE "MANDT" = '300' AND "BUKRS" = '3010' AND "BELNR" = '2009636653' AND "GJAHR" = 2009 ORDER
2,302	RFBLG	FETCH	1	100	
7	RFBLG	CLOSE	0	0	
4	RFBLG	REOPEN		0	SELECT WHERE "MANDT" = '300' AND "BUKRS" = '3010' AND "BELNR" = '2009968977' AND "GJAHR" = 2009 ORDER
2,403	RFBLG	FETCH	1	100	
8	RFBLG	CLOSE	0	0	

Figure 83: Financial report ST05

As expected from the STAT records in Figure 82, each call retrieves one row per call, and each call lasts about 2.5 ms. We explain the statement, and find that it matches the first four columns of the RFBLG~0 index.

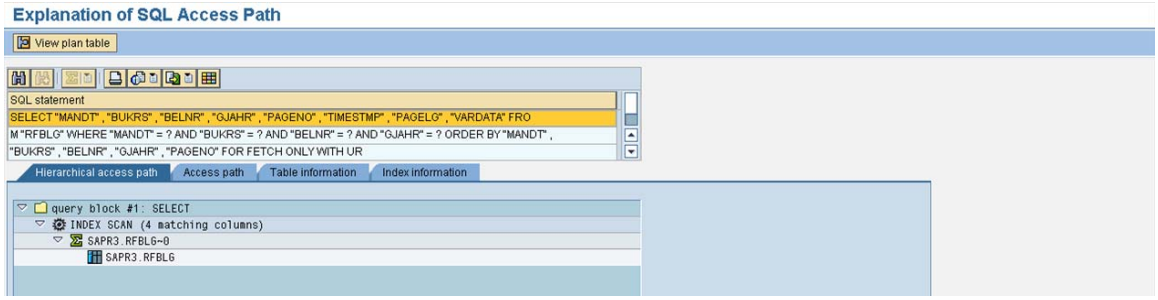


Figure 84: Financial report access path

Usually the ~0 index is also the DB2 clustering index, but as shown in Section 7.5.8, that can be changed. We check the index information in explain.

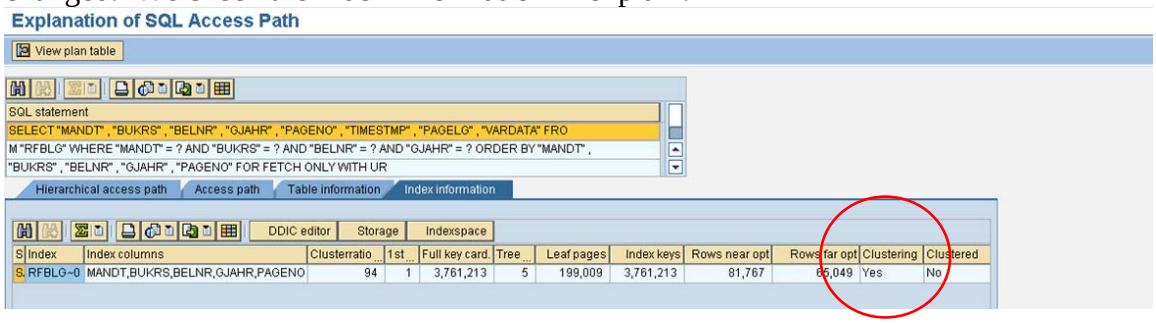


Figure 85: Financial report display index

Since RFBLG~0 is the clustering index, this means that for fixed values of MANDT, the BUKRS will be in ascending order, and for fixed values of MANDT and BUKRS (e.g. 300 and 3010 in the trace in Figure 83) the BELNR values will be in ascending order. But, we saw in the trace that the BELNR is not being fetched in ascending sequence.

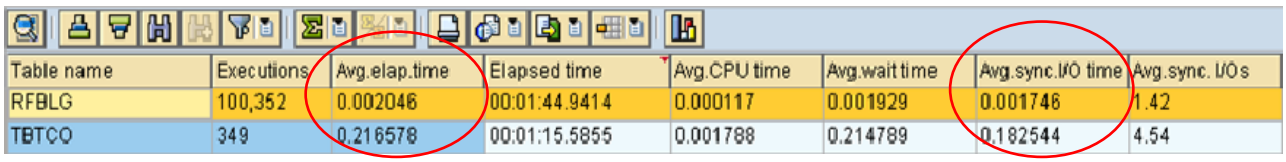


Table name	Executions	Avg.elap.time	Elapsed time	Avg.CPU time	Avg.wait time	Avg.sync.I/O time	Avg.sync. I/Os
RFBLG	100,352	0.002046	00:01:44.9414	0.000117	0.001929	0.001746	1.42
TBTCO	349	0.216578	00:01:15.5855	0.001788	0.214789	0.182544	4.54

Figure 86: Financial report ST04

We find the statement in the SQL cache as shown in Figure 86, and note that Synch I/O is about ¾ of the statement elapsed time.

When we access the rows without following the clustering sequence, this can increase I/O, since out-of-order access lessens the likelihood of finding the requested row in DB2 buffer pool memory.

In this case, we ask the programmers to review how the data is retrieved. For instance, if the keys for rows to be fetched from RFBLG are stored in an internal table, the internal table could be sorted to match the clustering sequence.

Compare the response time in Figure 86 with the following statement in Figure 87, where there is almost no wait time.

Table name	Executions	Avg.elap.time	Elapsed time	Avg.CPU time	Avg.wait time	Avg.sync.I/O time	Avg.sync. I/Os	TS scans	Sorts
RFBLG	14,471	0.000057	00:00:00.8293	0.000054	0.000003	0.000000	0.00	0	0

Figure 87: Financial report - RFBLG for ordered select

This statement in Figure 87 comes from a program doing the same select on RFBLG, but where the keys to be selected were ordered in the program internal table to match the clustering sequence.

Duration	Obj. name	Op.	Recs.	RC	Statement
886	RFBLG	FETCH	1	100	
5	RFBLG	CLOSE	0	0	
3	RFBLG	REOPEN		0	SELECT WHERE "MANDT" = '100' AND "BUKRS" = '1010' AND "BELNR" = '2008556578' AND "GJAHR" = 2008 ORDER BY "MANDT" , "BUKRS" , "BEL
788	RFBLG	FETCH	1	100	
5	RFBLG	CLOSE	0	0	
3	RFBLG	REOPEN		0	SELECT WHERE "MANDT" = '100' AND "BUKRS" = '1010' AND "BELNR" = '2008556579' AND "GJAHR" = 2008 ORDER BY "MANDT" , "BUKRS" , "BEL
889	RFBLG	FETCH	1	100	
5	RFBLG	CLOSE	0	0	
3	RFBLG	REOPEN		0	SELECT WHERE "MANDT" = '100' AND "BUKRS" = '1010' AND "BELNR" = '2008556580' AND "GJAHR" = 2008 ORDER BY "MANDT" , "BUKRS" , "BEL
681	RFBLG	FETCH	1	100	
5	RFBLG	CLOSE	0	0	

Figure 88: Financial report ordered rows

Since each select is retrieving a single row, it may also be possible to further optimize this program with array select. But regardless of whether array or single row selects are used, *retrieving the rows in a way that matches the clustering sequence will generally help reduce DB request time.*

7.5.10. Analysis of multiple dialog steps with transaction SQLR

The SQLR transaction (also named SQLR0001 or /SQLR/0001 program, depending on SAP version) can be used to merge an ST05 trace with STAT records, in order to match SQL operations to their dialog step, as one would need to do when analyzing a trace from a business process made up of several dialog steps.

Since one goal in improving performance is to find problems that have a significant affect on performance, by comparing the length of the SQL operations with the amount of database request time for the dialog step, one can estimate the possible improvement available.

In this example, there was a complaint about the performance of VA01. The ST05 trace has already been made while a user ran all the dialog steps of the slow the VA01 transaction. Now we are going to re-process the ST05 trace with SQLR.

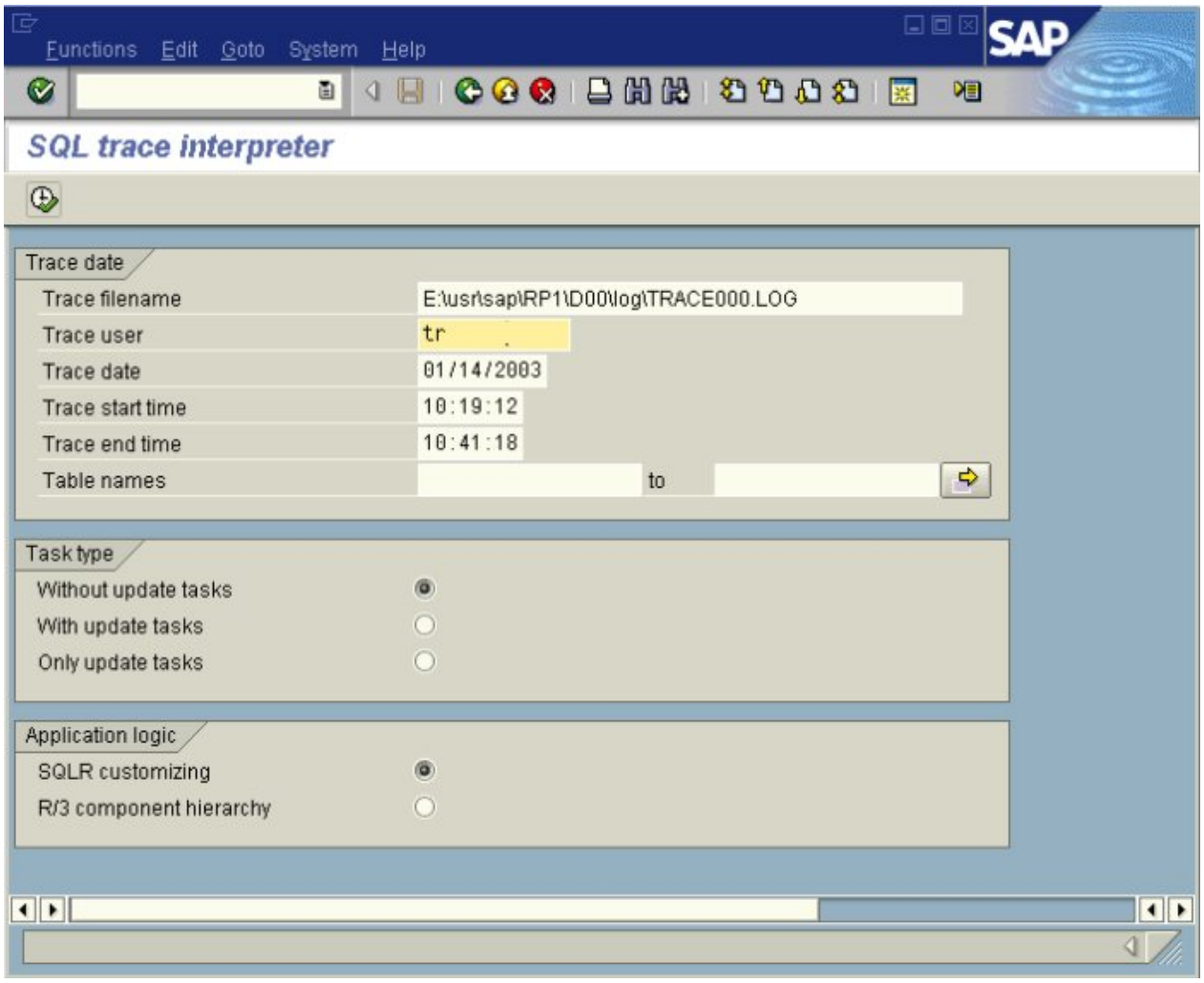


Figure 89: SQLR initial screen

Just as with ST05 trace list function, in Figure 89 one fills in the start and end times, along with the name of the user to be analyzed, selected tables, etc.

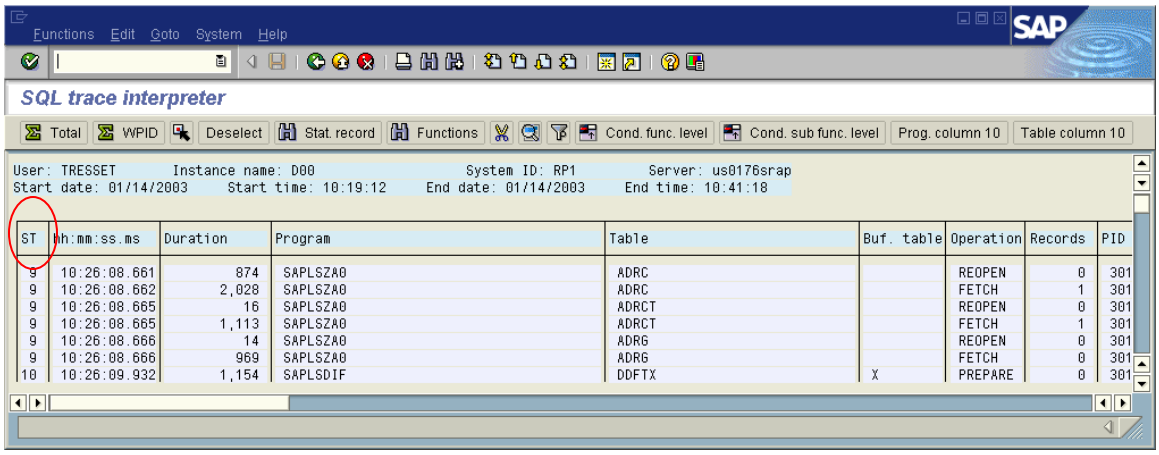


Figure 90: SQLR formatted trace

The formatted trace is similar to an ST05 trace, containing duration in microseconds, table name, and time of day. In addition, the STAT record is assigned a number, which is displayed in the "ST" column.

Press the "stat. record" button on Figure 90 to display the STAT records that cover the ST05 trace.

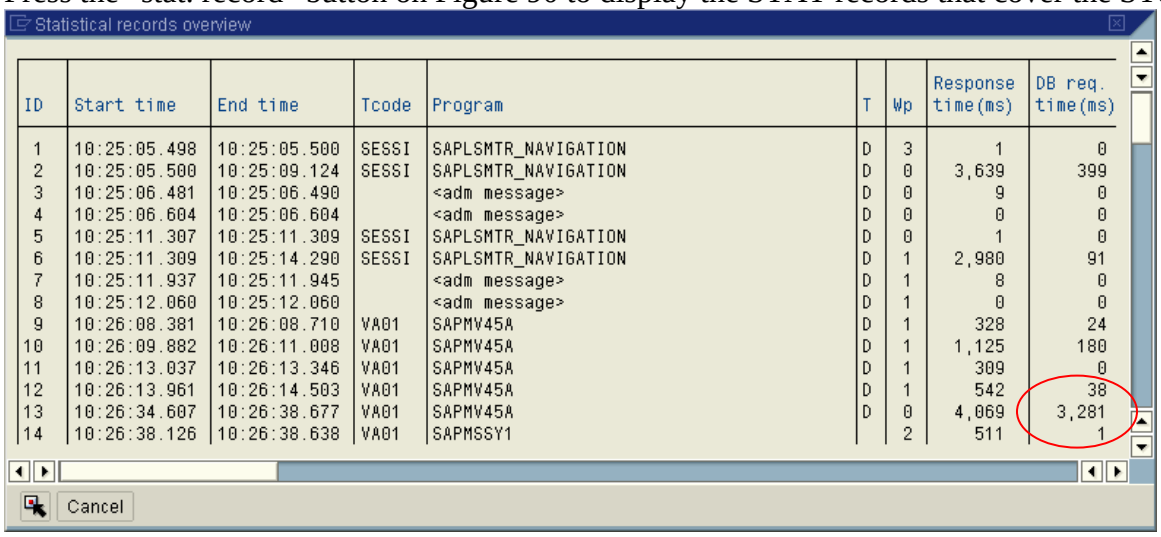


Figure 91: SQLR STAT record display

In this example, we would review the formatted trace in Figure 90 for SQL related to dialog step ID 13, since that is the dialog step with the longest DB request time, and thus the greatest opportunity for improvement.

7.6. Batch

7.6.1. Analysis process for batch

As with transaction analysis, the two main tools are ST05 and SE30. Use ST05 to find inefficient SQL, and determine what activities take the most database request time. SE30 is used to profile the time running on the application server.

Since batch jobs are long running, one can gather additional information on the job, using ST04 thread information and ST06 CPU information, to get an end-to-end profile of elapsed time that includes CPU on the application server, CPU on the database server, delay on the database server, and "overall not accounted" time, which includes network time and STAT "missing" time. The sources of delay in DB2 are discussed in more detail in section 8.1.

If the job is long running and the statistics for a counter have wrapped, then the STAT records are not helpful in filtering the problem source – whether it is excessive database request time, etc. Analysis of the job while it is running is required.

Even if the STAT counters have wrapped, the data recorded in stat/tabrec may still be valid, so *stat/tabrec* and *rsdb/stattime* can be helpful in gathering data about performance issues in long running batch jobs. These two SAP parameters should not be enabled all the time, just during detailed problem analysis.

7.6.2. Using SE30 to determine cause of excessive CPU use

SE30 can be used to determine the components of response time for batch jobs or transactions. When the problem is high CPU use on the application server, SE30 will show which ABAP routines take the most time. One can then focus on these problem areas. When running SE30 for a transaction, start the transaction from within SE30. When tracing a batch job, trace the job in parallel with SE30, as in this example.

For this example, assume that we have already reviewed the STAT/STAD or ST03 statistics for this next job, and have determined that the majority of elapsed time is CPU on the application server.

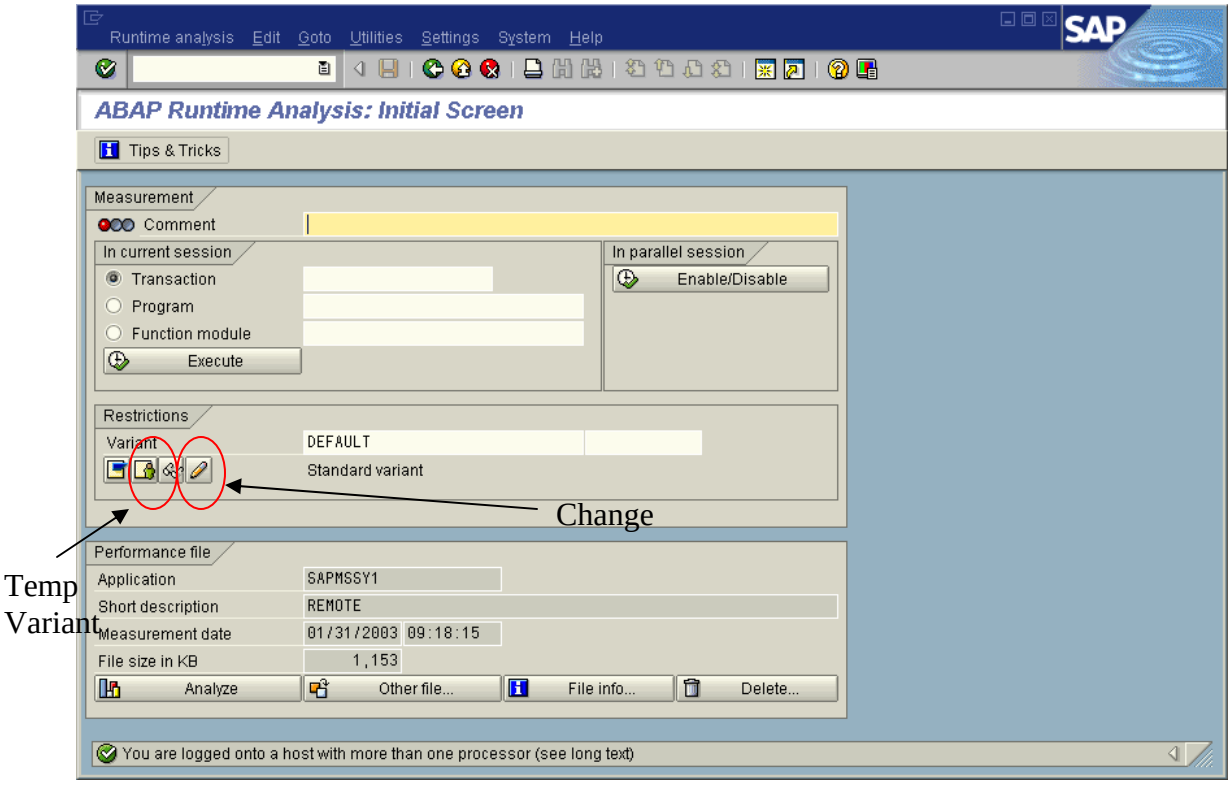


Figure 92: SE30 initial screen

To run SE30 using customized traces, select the TMP variant, then press the change button.

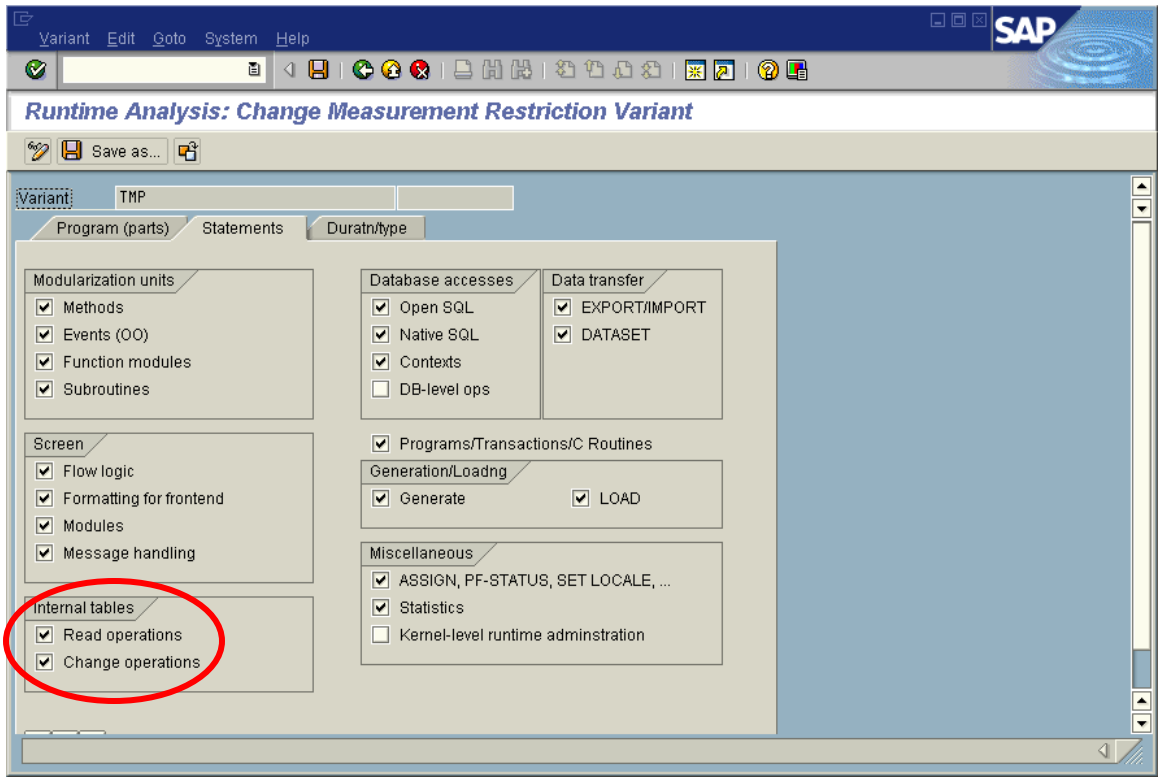


Figure 93: SE30 Statement options

To trace activity on internal tables (this cannot be traced with ST05) select the Read and Change operations at the bottom left. This can make the trace grow very quickly. Inefficient read from internal tables is a common scalability problem in ABAP programming.

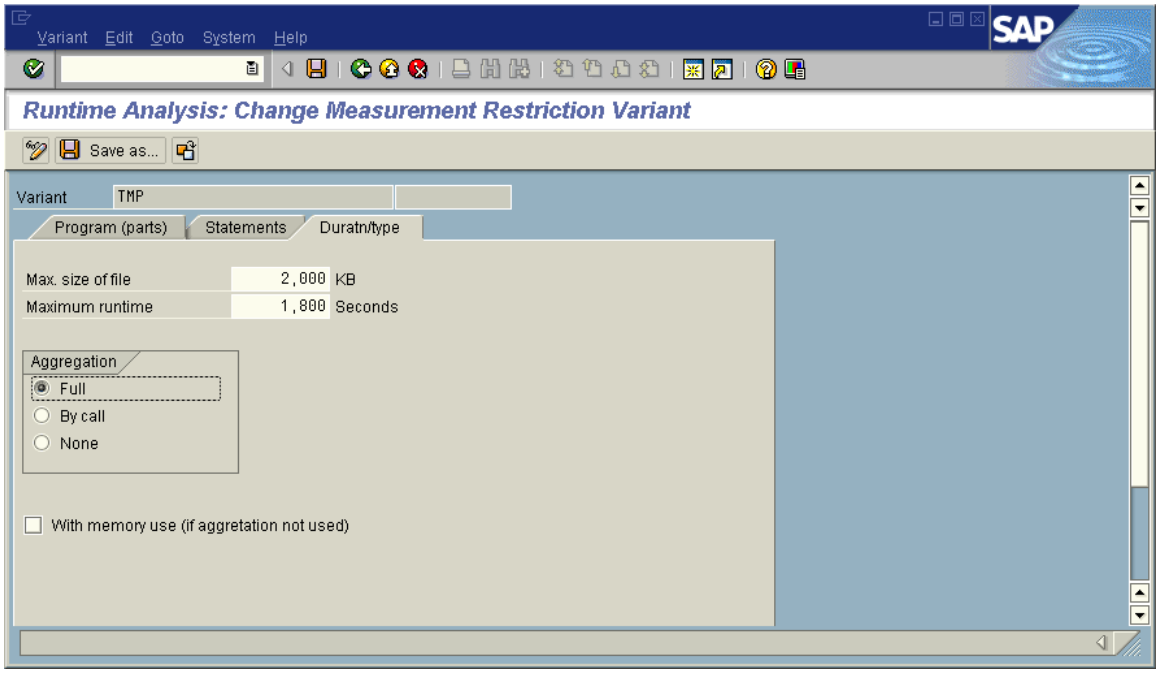


Figure 94: SE30 set duration and aggregation

Limits can be placed on the trace time, or size of trace file.

The aggregation radio button controls how the trace is summarized. If “None” is selected, then you will be able to see the response time of each DB operation, each internal table read, etc. Full aggregation summarizes all the calls into a single total time. One can calculate the average time from count and total time, but one cannot see if there is a skew in the call times.

Save and go back to the initial SE30 screen (Figure 92) and press the “Enable/Disable” button under “in parallel session”.

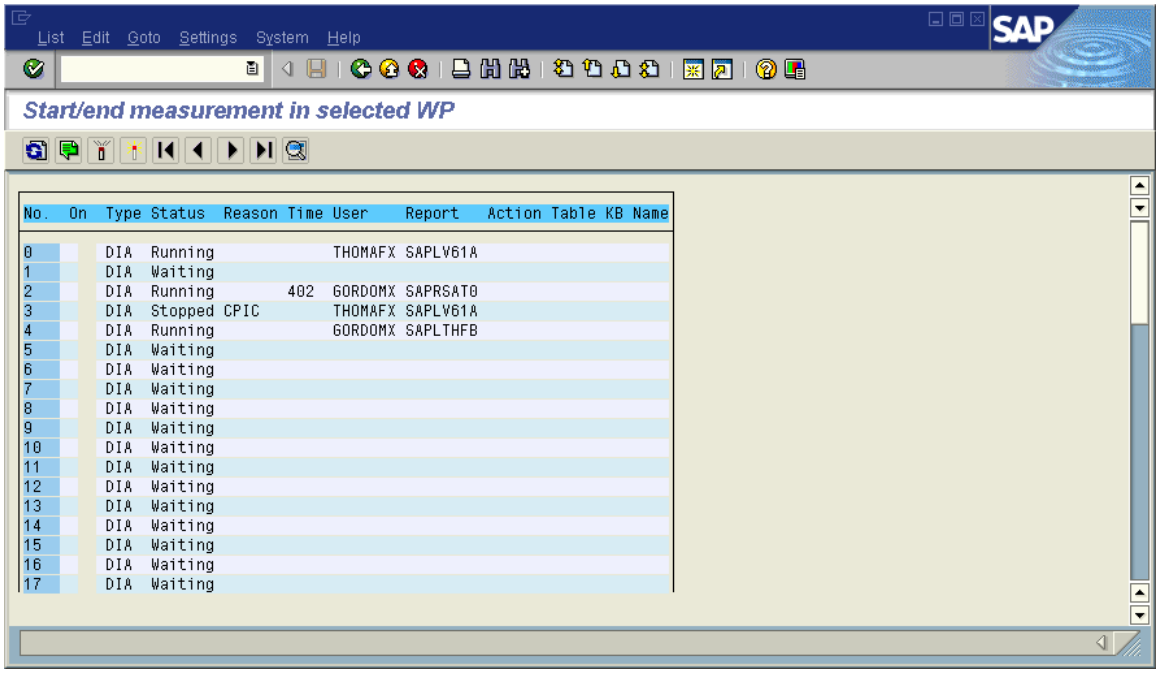


Figure 95: SE30 process list

Select the process to be traced, and press the activate button (4th button from left).

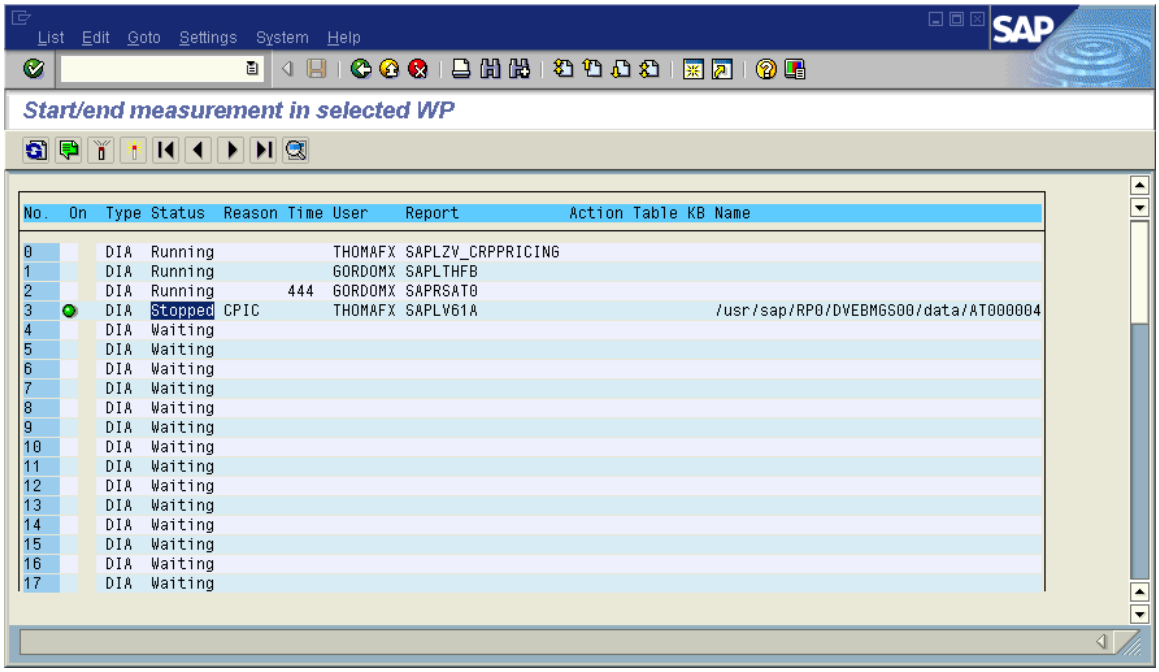


Figure 96: SE30 activated trace

Run the trace for a while. Then deactivate it. This brings you back to the main SE30 panel again. Press Analyze.

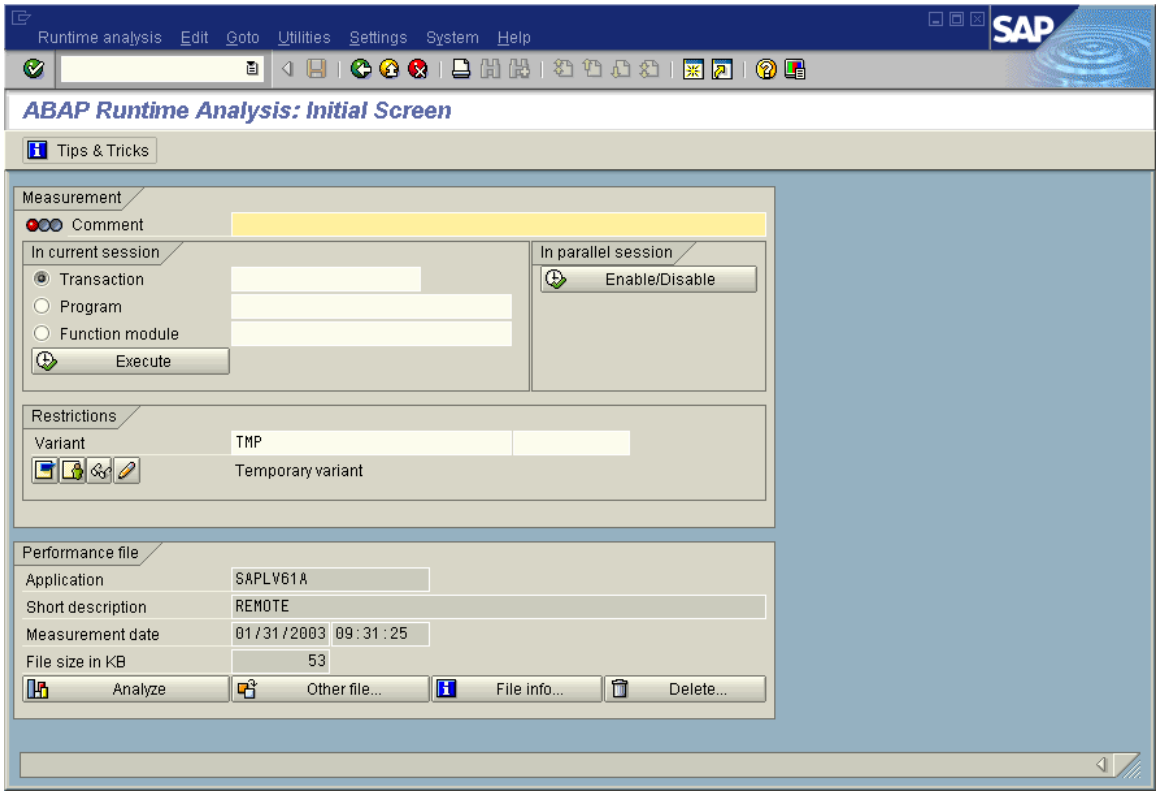


Figure 97: SE30 analyze

After the analysis is done, you get an overview of time during the trace interval.

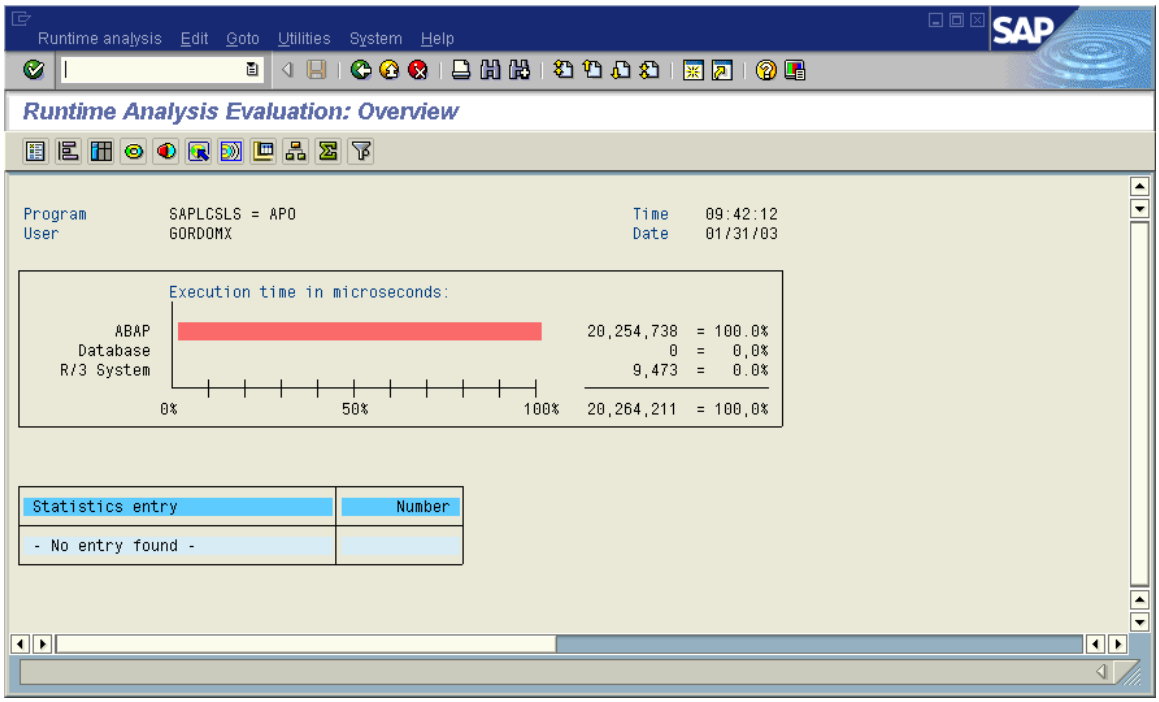
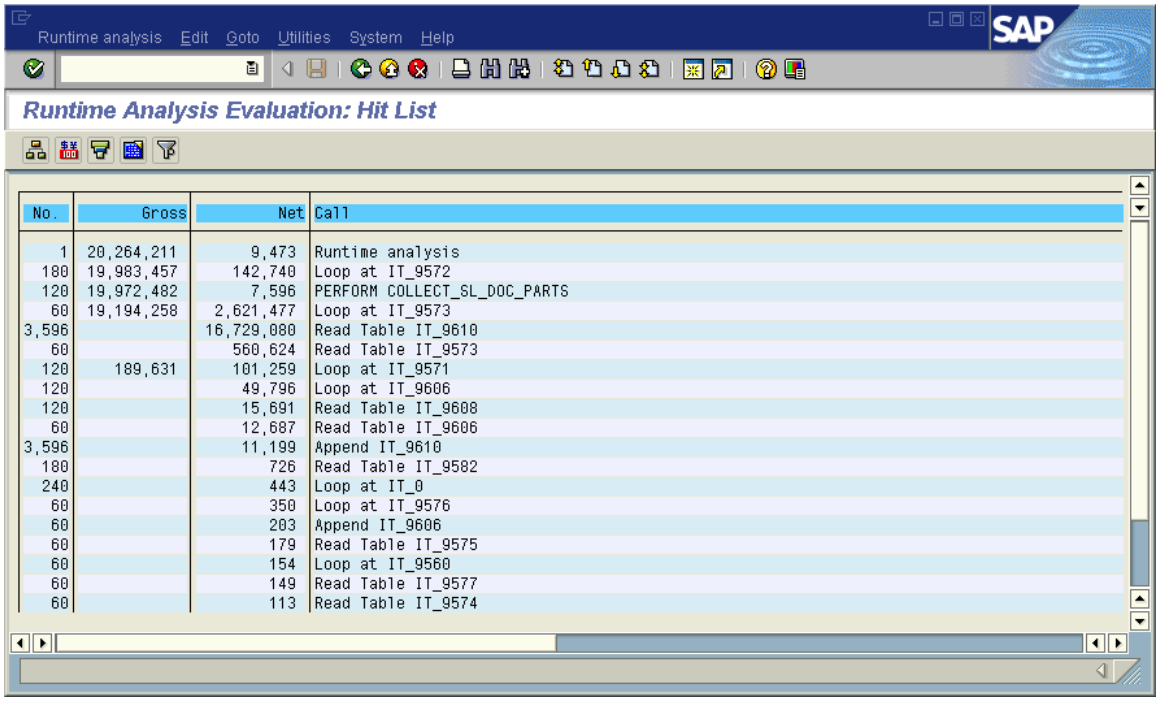


Figure 98: SE30 Runtime Analysis Overview

In Figure 98, 100% of 20 seconds elapsed time (units are micro-seconds) is in ABAP time. This contains CPU processing, as well as RFC and other time on the application server. Press the 'Hit List" button on the top left.



No.	Gross	Net	Call
1	20,264,211	9,473	Runtime analysis
180	19,983,457	142,740	Loop at IT_9572
120	19,972,482	7,596	PERFORM COLLECT_SL_DOC_PARTS
60	19,194,258	2,621,477	Loop at IT_9573
3,596		16,729,080	Read Table IT_9610
60		560,624	Read Table IT_9573
120	189,631	101,259	Loop at IT_9571
120		49,796	Loop at IT_9606
120		15,691	Read Table IT_9608
60		12,687	Read Table IT_9606
3,596		11,199	Append IT_9610
180		726	Read Table IT_9582
240		443	Loop at IT_0
60		350	Loop at IT_9576
60		203	Append IT_9606
60		179	Read Table IT_9575
60		154	Loop at IT_9560
60		149	Read Table IT_9577
60		113	Read Table IT_9574

Figure 99: SE30 hit list

Then sort by Net time to determine the routine where the most time is spent. This is net as in Net/Gross, not network.

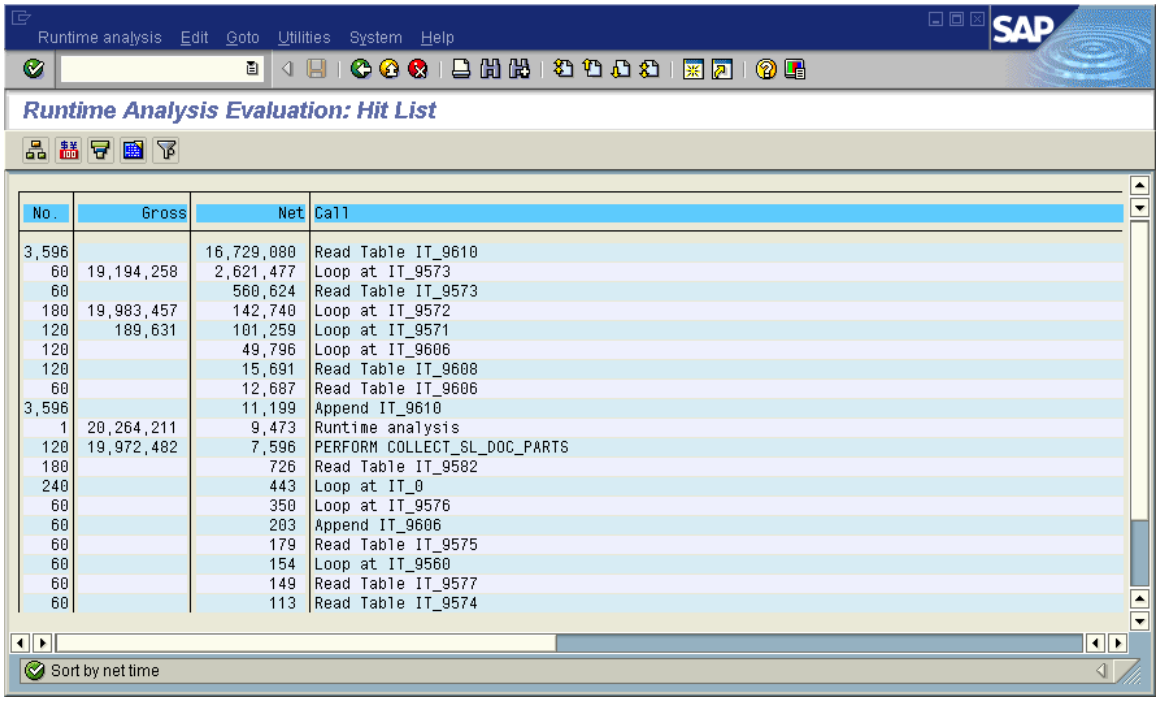


Figure 100: SE30 sorted by Net time

Drill in to the long-running Read Table.

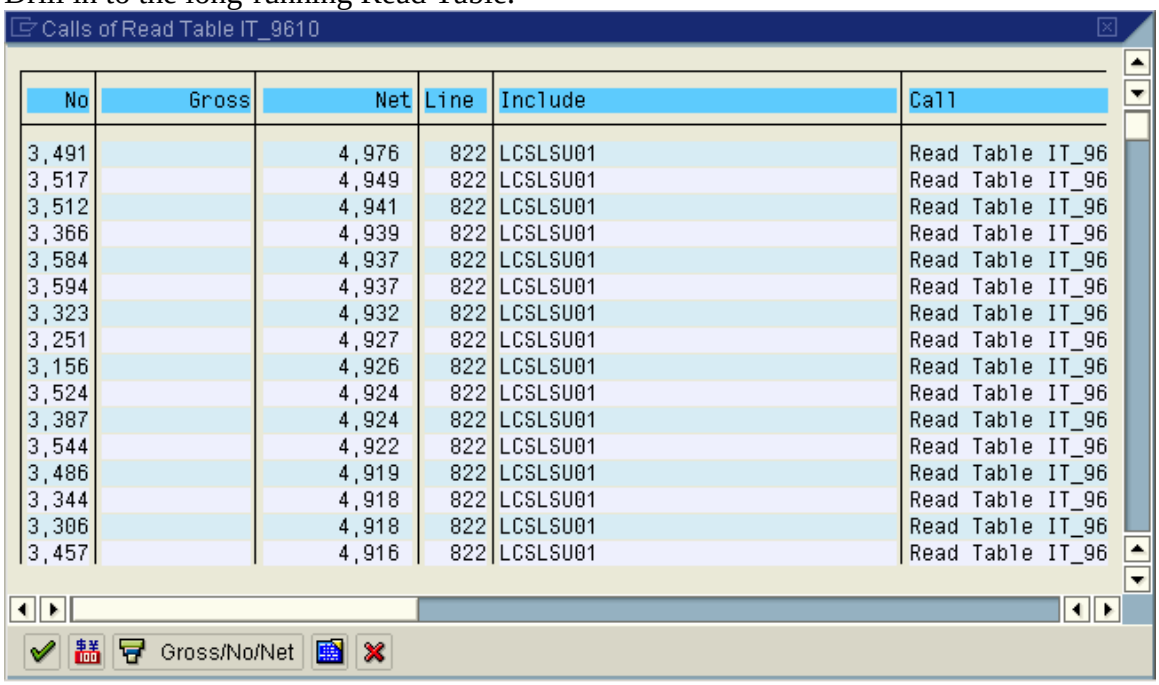


Figure 101: SE30 long reads from internal table

Each read from an internal table is nearly 5 ms., which is very long. In general, internal table read will be just a few microseconds. Press “display source code” on the bottom row.

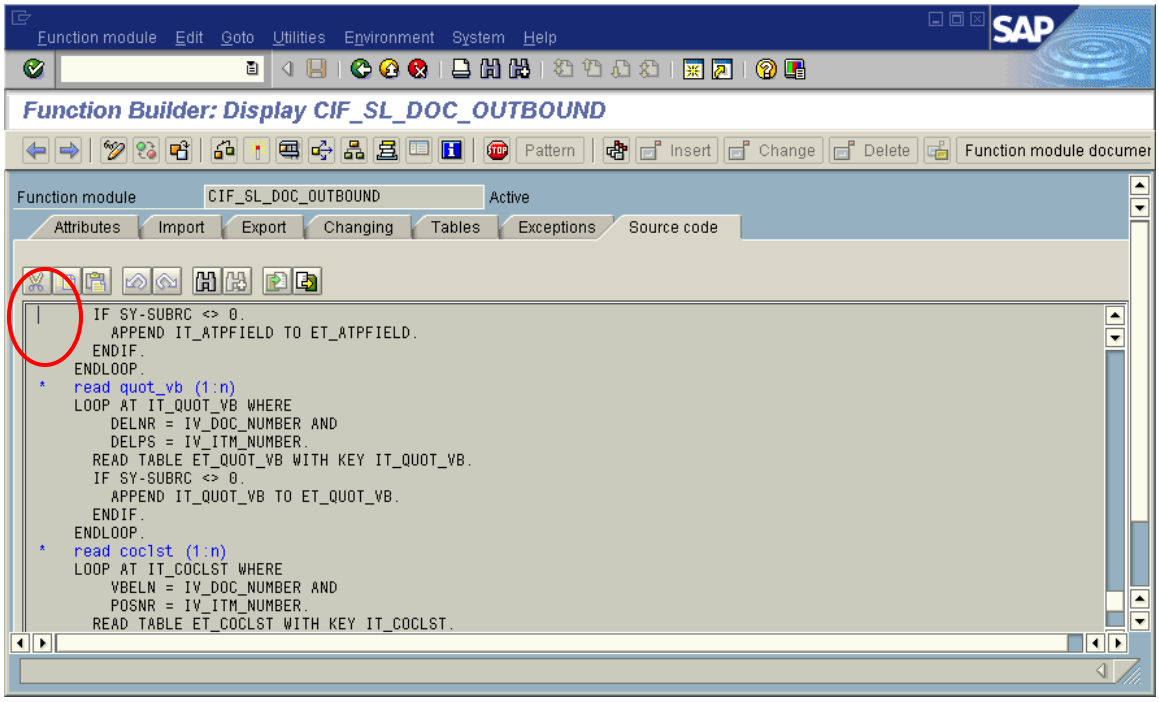


Figure 102: SE30 cursor positioned after offending line

The cursor will be positioned after the slow statement. Page up to see the statement.

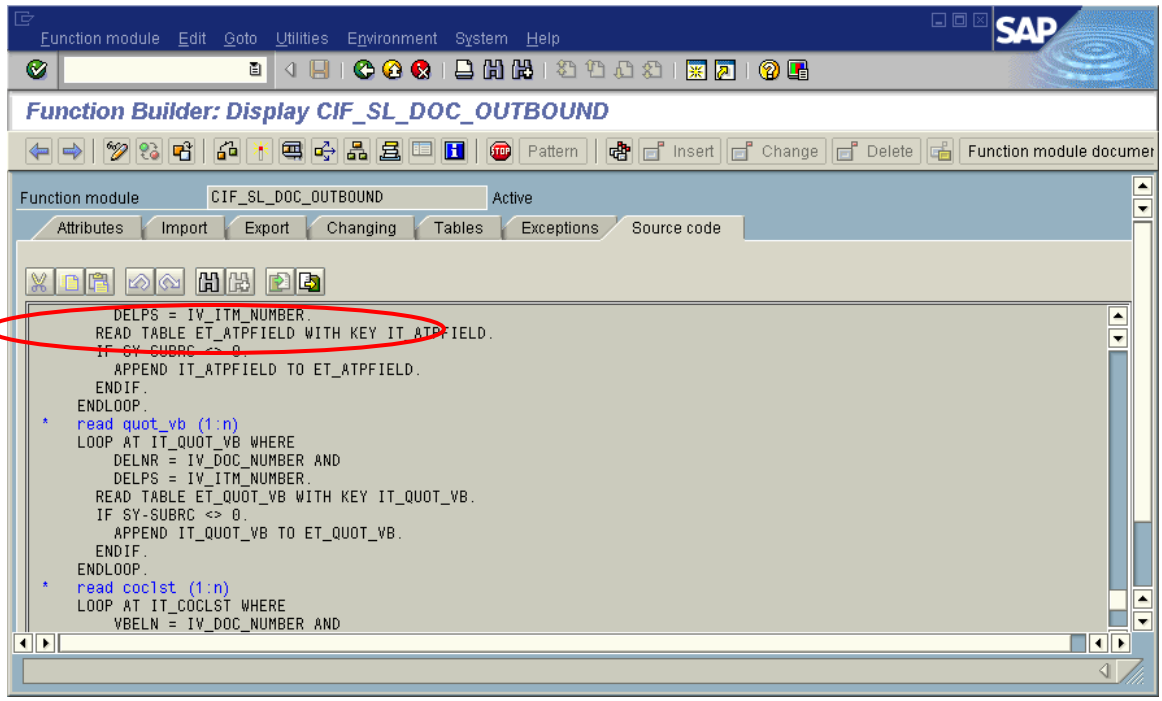


Figure 103: SE30 slow statement found

Now we have found the statement, what is the problem? Since this is SAP code, we could open an OSS message. SE30 has a Tips and tricks section, with common problems. This is the 'linear search vs binary search' example.

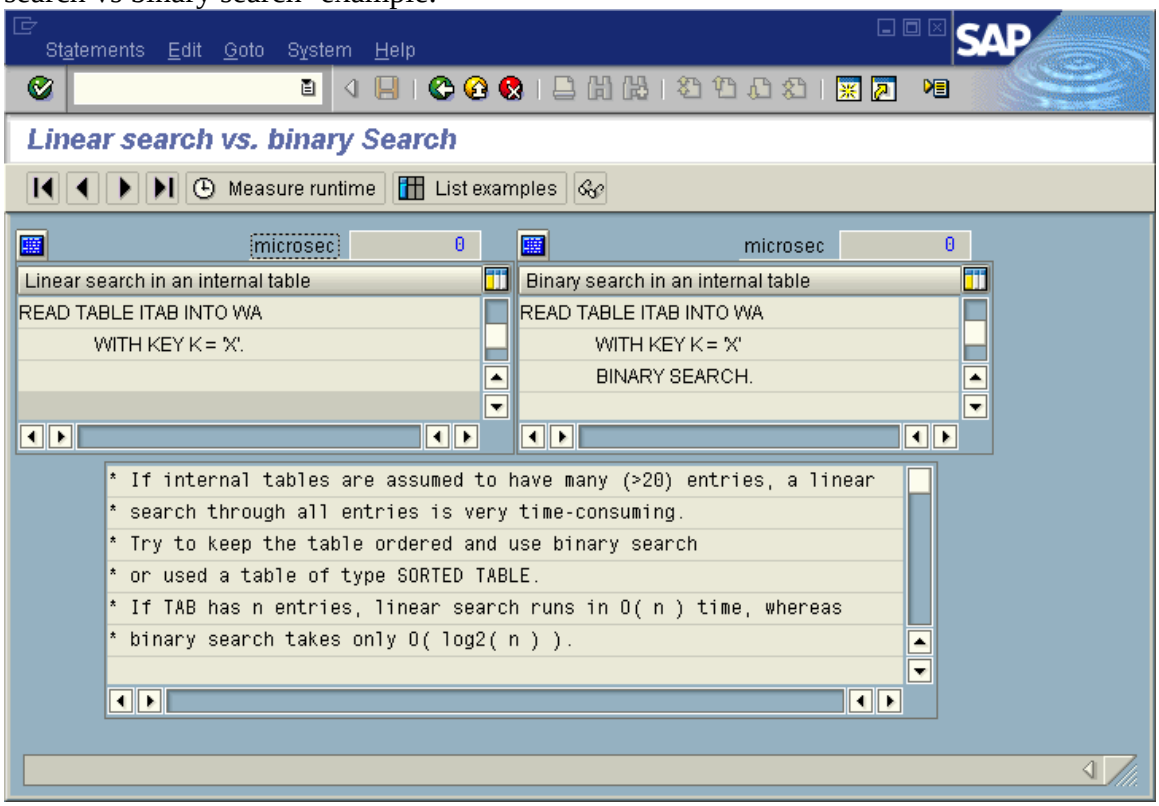


Figure 104: SE30 Tips and tricks – linear search vs binary search

So, in our example above, the fix (from SAP) would be to change the ABAP to use the "BINARY SEARCH" option on the "READ TABLE" operation.

7.6.3. Sample of SQL cycle analysis in batch

When tracing a batch job, it is important to find the cyclical behavior of the job, so that analysis of the SQL can be performed through at least a full cycle. Analyzing and aggregating several cycles is preferable, in order to average out the impact of transient conditions.

Start, list, and summarize an ST05 trace of the batch job. In the summarized trace, look for the markers of a cycle. First, look for the "commit work" statements. There may be several cycles within a single commit work, there may also be one cycle per commit work, or there may be several commit works per cycle. In this case, choose the update to TST01 as the marker for a cycle. Select the starting line, then "edit > select beginning".

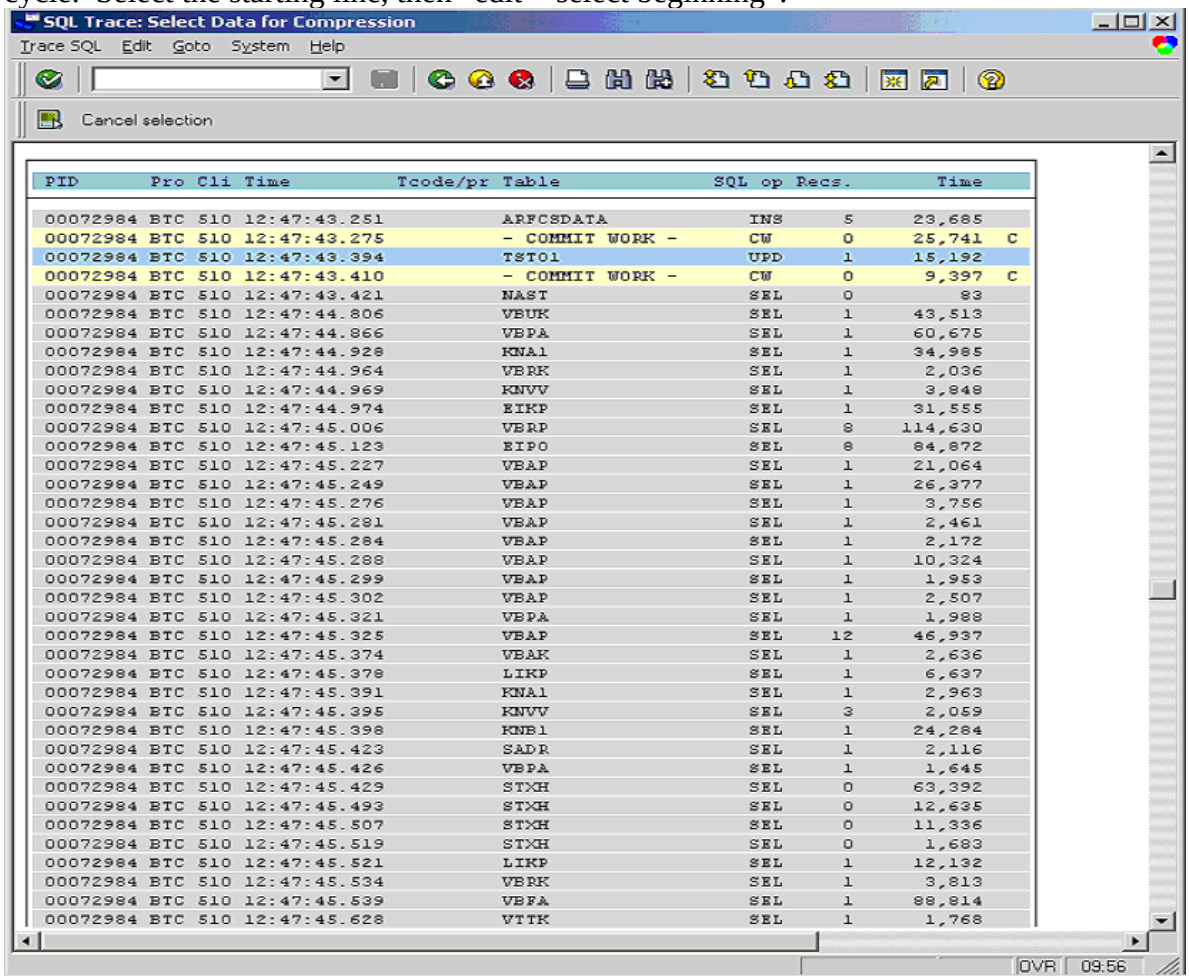


Figure 105: TST01 - Select starting point in summarized ST05 trace

Then, use “find” to locate additional markers of the cycle – TST01.

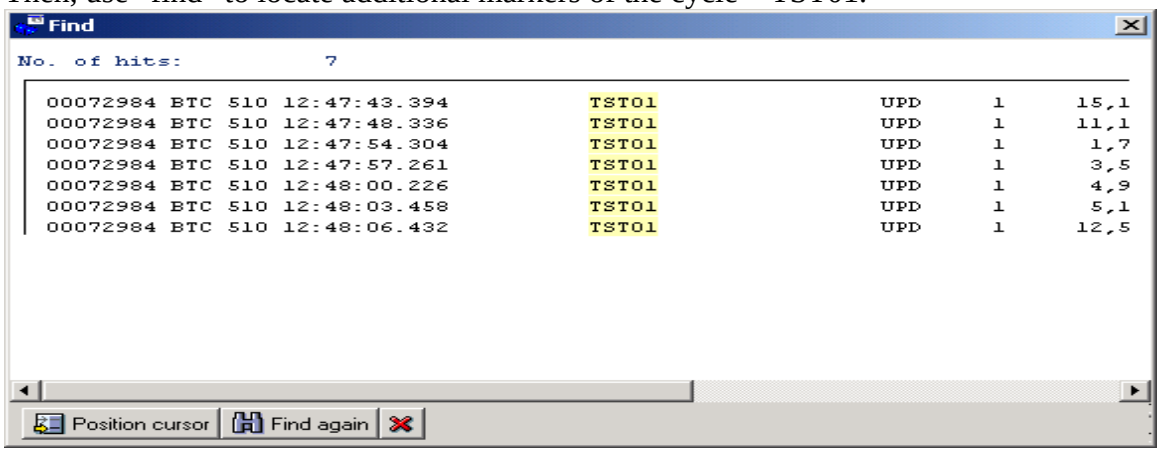


Figure 106: ST05 cycle marker example

Select the end of the cycle: "edit > select end". Here, the selected range ends with the ARFDSDATA insert before the TST01 update.

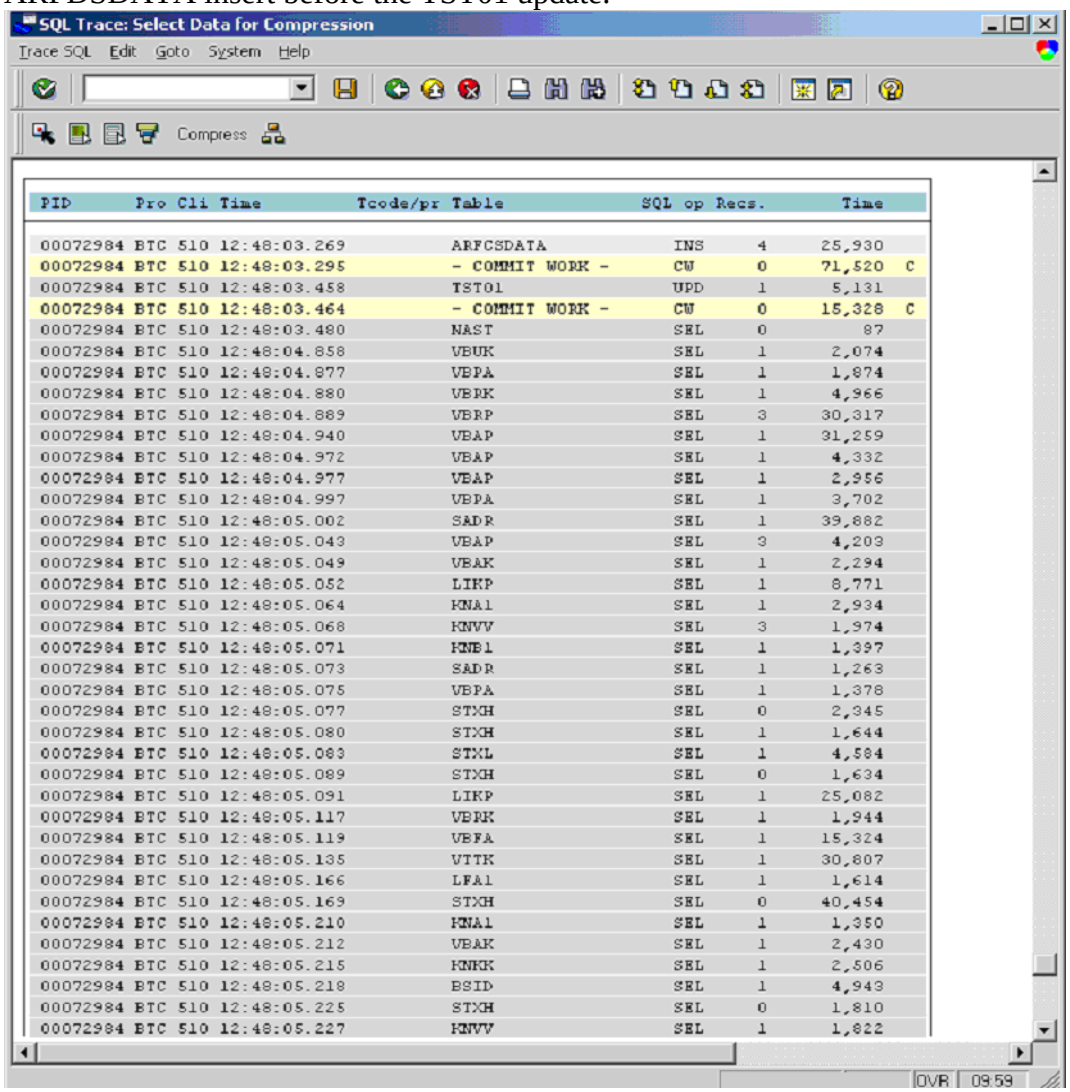


Figure 107: ST05 select end

Use "edit > set tcode" to put an identifier into the trace. ST05 does not care what is entered, so make it something that will help to interpret the result, if you look at this a month from now.

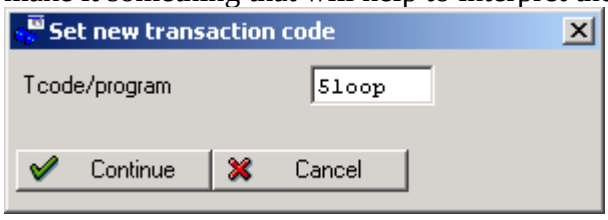
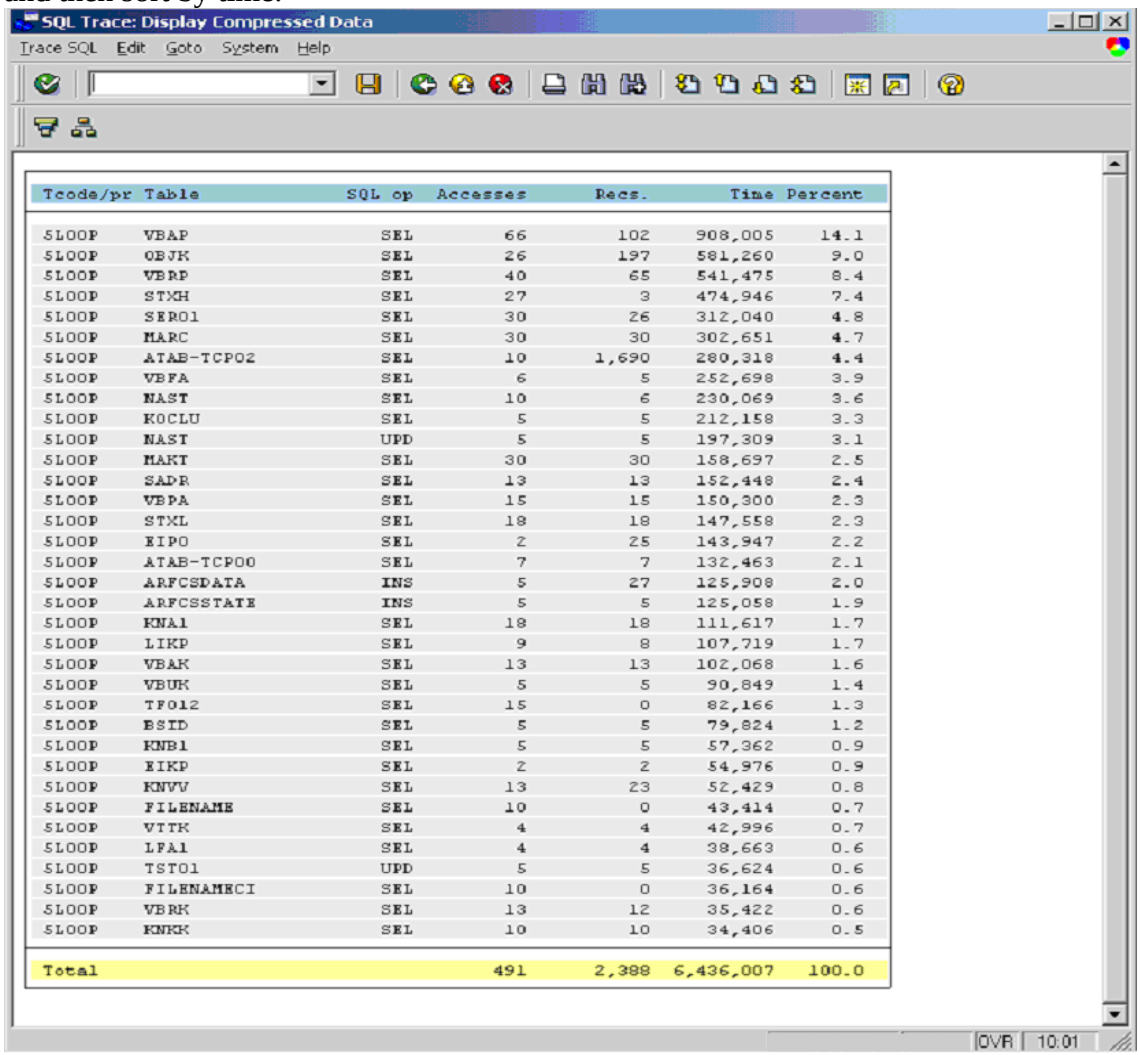


Figure 108: ST05 set tcode

Press the summarize button (also labeled compress in some SAP versions) to compress the trace, and then sort by time.



Tcode/pr	Table	SQL op	Accesses	Recs.	Time	Percent
SLOOP	VBAP	SEL	66	102	908,005	14.1
SLOOP	OBJK	SEL	26	197	581,260	9.0
SLOOP	VBRP	SEL	40	65	541,475	8.4
SLOOP	STXH	SEL	27	3	474,946	7.4
SLOOP	SER01	SEL	30	26	312,040	4.8
SLOOP	MARC	SEL	30	30	302,651	4.7
SLOOP	ATAB-TCP02	SEL	10	1,690	280,318	4.4
SLOOP	VBFA	SEL	6	5	252,698	3.9
SLOOP	NAST	SEL	10	6	230,069	3.6
SLOOP	KOCLU	SEL	5	5	212,158	3.3
SLOOP	NAST	UPD	5	5	197,309	3.1
SLOOP	MAKT	SEL	30	30	158,697	2.5
SLOOP	SADR	SEL	13	13	152,448	2.4
SLOOP	VBPA	SEL	15	15	150,300	2.3
SLOOP	STXL	SEL	18	18	147,558	2.3
SLOOP	EIPO	SEL	2	25	143,947	2.2
SLOOP	ATAB-TCP00	SEL	7	7	132,463	2.1
SLOOP	ARFCSDATA	INS	5	27	125,908	2.0
SLOOP	ARFCSSTATE	INS	5	5	125,058	1.9
SLOOP	KNA1	SEL	18	18	111,617	1.7
SLOOP	LIKP	SEL	9	8	107,719	1.7
SLOOP	VBAK	SEL	13	13	102,068	1.6
SLOOP	VBUK	SEL	5	5	90,849	1.4
SLOOP	TF012	SEL	15	0	82,166	1.3
SLOOP	BSID	SEL	5	5	79,824	1.2
SLOOP	KNB1	SEL	5	5	57,362	0.9
SLOOP	EIKP	SEL	2	2	54,976	0.9
SLOOP	KNVV	SEL	13	23	52,429	0.8
SLOOP	FILENAME	SEL	10	0	43,414	0.7
SLOOP	VTTK	SEL	4	4	42,996	0.7
SLOOP	LFA1	SEL	4	4	38,663	0.6
SLOOP	TST01	UPD	5	5	36,624	0.6
SLOOP	FILENAMECI	SEL	10	0	36,164	0.6
SLOOP	VBRK	SEL	13	12	35,422	0.6
SLOOP	KNKK	SEL	10	10	34,406	0.5
Total			491	2,388	6,436,007	100.0

Figure 109: ST05 summarized and sorted

Now evaluate the summary and look for slow tables. The time units are microseconds. At the top of the figure, 102 VBAP rows are read in 908 ms (908,005 microseconds), for an average of almost 9 ms per row, which is a bit slow.

Since the top table is only 14% of DB time, unlike the transaction trace in section 7.5.2, there is not a big problem on any of the tables. The purpose of this section is to demonstrate a process for analyzing SQL for batch jobs.

If there were a problem, use ST05 to explain long running SQL statements, check whether a better index is available, of if a new index might be needed, or if the program should be changed.

8. Check for inefficient use of DB resources

8.1. DB2 accounting data – delay analysis

DB2 accounting data can be used to determine where time is spent processing in DB2. Time is gathered on each DB2 thread, and is broken down into "class 1", "class 2", and "class 3" time.

- Class 3 suspend time is when the thread is suspended, and waiting for a DB2 action (commit, I/O, row lock, etc).
- Class 2 Elapsed time is when DB2 is processing a request – it contains the Class 3 delay time, as well as CPU time in DB2 (class 2 CPU time).
- Class 1 CPU is time that a thread is processing SQL from SAP – it contains class 2 CPU, plus time processing on S/390 outside DB2, e.g. time in the ICLI.
- Class 1 elapsed time is the time a thread is allocated. This is not a useful performance metric with SAP.
- Not attributed time = *Class 2 elapsed time – Class 2 CPU time – Class 3 suspension time*. It is the time that is left when all the time that DB2 can account for is removed from DB2 elapsed time. Not attributed time happens when DB2 considers a thread to be runnable, but the thread is not running. This might be caused by a CPU overload on the DB server, a paging problem on the DB server, etc. In some versions of SAP, this is reported as "Other" time in ST04 "times".

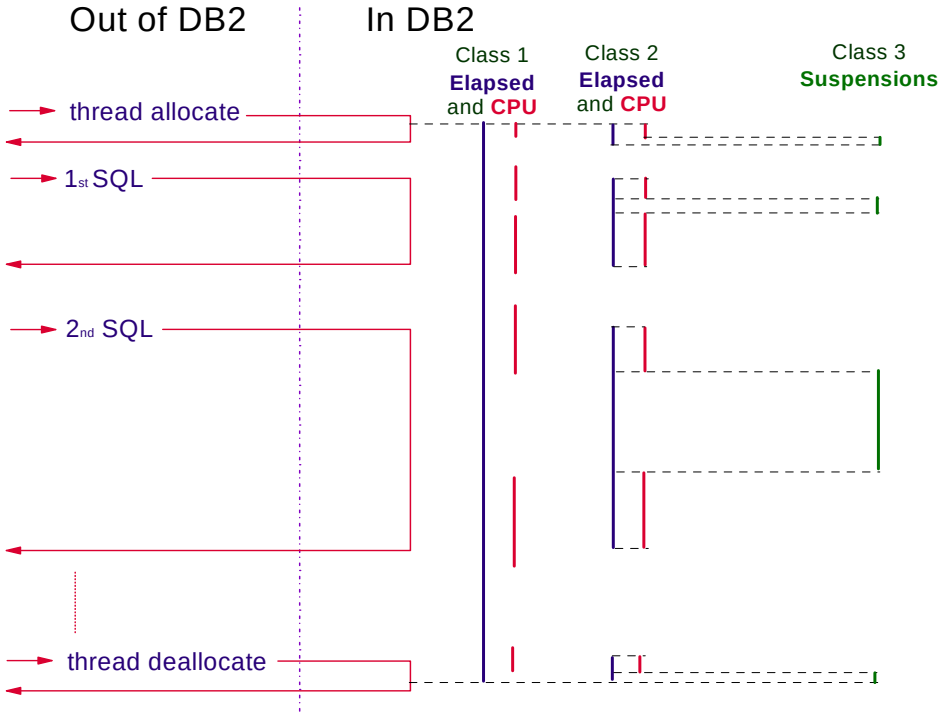


Figure 110: DB2 time categories

At a system-wide level, the ST04 "global times" function (or DB2PM accounting report) can be used to display the main sources of delay in DB2. Since these delays are generally a symptom of another problem

(e.g. inefficient SQL causes excessive I/O which causes high I/O delay in DB2), ST04 “times” is best used to get an overview of the system performance in DB2, and to get a feeling for the possible gains which can be achieved from tuning.

ST04 (DBACOCKPIT) statement cache display can be used to examine the delays of individual statements.

ST04 “global times” data is calculated from active threads. Since SAP DB2 threads may terminate and restart over the course of a day, one should evaluate ST04 “times” at different times of the day, or aggregate the thread accounting history with DB2PM, in order to see the overall impact of delays in DB2. Long running threads, such as threads for monitoring programs, can skew the “global times” data. Check the ST04 thread display, and sort the threads by time, to determine if ther are long-running threads that are skewing the “global times” data.

A ratio of about 50% delay in DB2 and 50% CPU in DB2, is very good for a productive SAP system. If the inefficient SQL has been removed, and the DB2 subystem is achieving 50% CPU in “global times”, then there is probably little opportunity for improvement. Ratios of up to 75% delay and 25% CPU are very often seen in normal productive systems. If, however, the system has a ratio of 80% (or higher) delay to 20% (or lower) CPU, and you want to improve overall DB server performance, then some additional analysis can show if this is a sign of a system-wide performance problem (inefficient SQL, slow I/O performance, etc) that needs to be addressed. In general, the bigger the delay, the larger the opportunity for improvement, and the easier it is to get improvement.

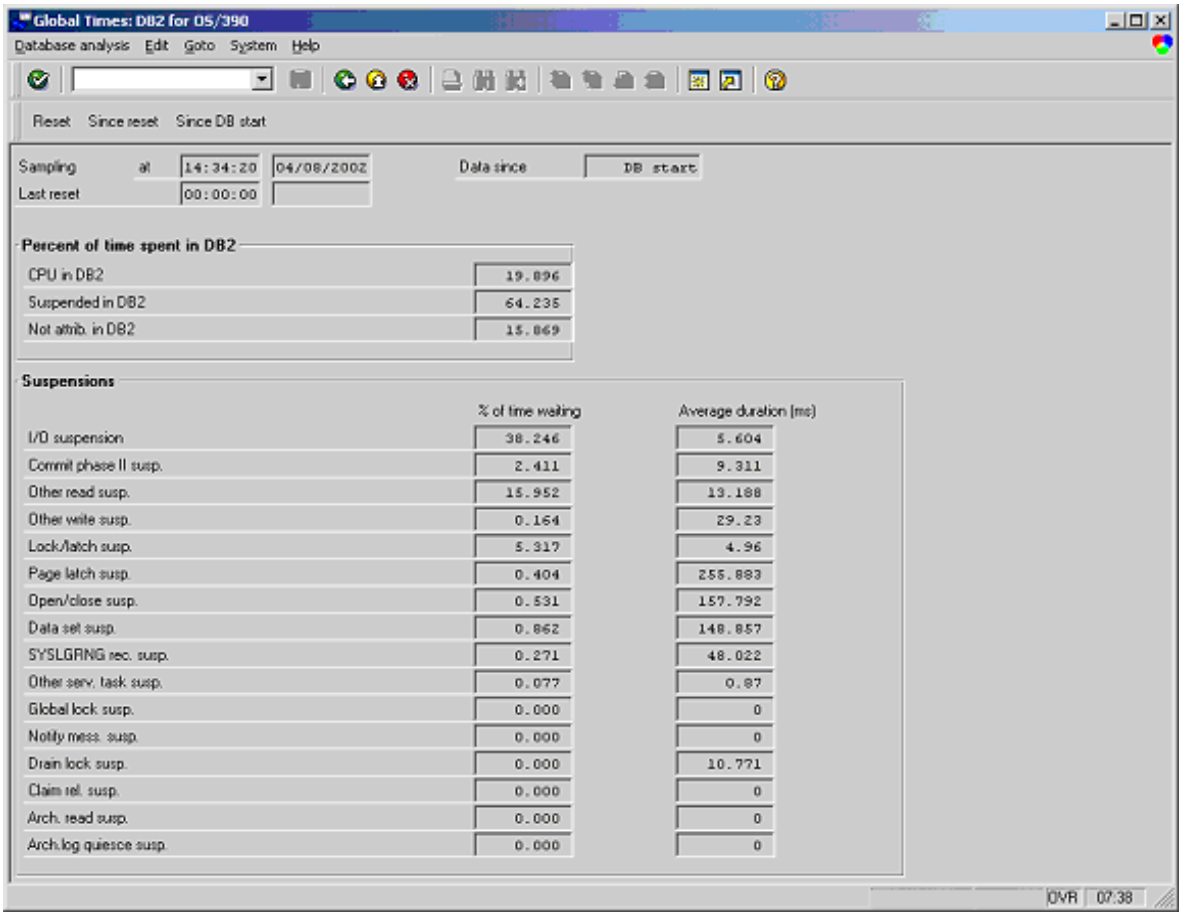


Figure 111: ST04 global times

8.1.1. Components of DB2 delay

The DB2 administration guide (SC26-9003) describes the "class 3" delays in detail. The most common delays seen with SAP systems are:

- I/O suspension, which is synchronous read by a DB2 thread. Synchronous I/O is executed by the thread running application SQL.
- Phase II commit, which is wait for commit processing, which includes logging.
- Other read suspension, which is wait for prefetch read, or wait for synchronous read by another thread. Unlike synchronous I/O, prefetch is not done by the DB2 thread running application SQL, but by prefetch processes.
- Other write suspension, where the DB2 thread is waiting for a DB2 page to be written.
- Lock/Latch suspension, which is logical (row level) lock suspension, as well as DB2 internal latch delay, and latch suspensions in IRLM.
- Page latch suspension, which is DB2 page contention. Since only one thread at a time can be changing a page, if several different threads simultaneously try to change different rows in the same page, there will be page latch contention. Also, in tables with very high insert activity, DB2 space mapping pages may have contention.
- Open/Close suspension, which is dataset open and close.

- Global lock suspension, which is data sharing locking suspension.

8.1.2. Key indicators in DB2 Times

- **Suspend in DB2 high** - (*class 3 / Class 2 elapsed*) – when this is high (e.g. over 75-80%), DB2 execution is often blocked while DB2 waits for events such as I/O, locks, etc.
- **Not attributed (or "Other") high** – when this is high (e.g. over 20-25%), DB2 execution is blocked due to an operating system issue such as CPU overload, workload prioritization, paging, etc.
- **CPU time high** - (*Class 2 CPU / Class 2 elapsed*) – if this is high (e.g. over 60-70%), there may be problems with inefficient SQL, such as tablescans on moderate sized memory resident tables. It may be a sign of a well-tuned system (high hit rates, short suspend times), though in general, it is unusual to see a system with Class 2 CPU greater than Class 2 Elapsed.
- **Length of individual suspensions** – long average duration for I/O suspension, other read I/O, other write I/O, and commit can be indicators of I/O performance problems.

8.1.3. Actions to take for DB2 times indicators

- High CPU time:
  - Look at ST04 statement cache for inefficient SQL.
  - Check DB2 trace settings
- High "I/O suspension" time (also called "synchronous read and write"):
  - Generally the largest source of delay in DB2
  - Check for inefficient SQL, see section 0. Inefficient SQL will reference more pages than necessary, which makes it difficult for DB2 to keep necessary data in bufferpools and hiperpools.
  - After checking SQL, if average suspension time is good (e.g. < 10 ms) and total I/O suspension time is high, then the DB2 bufferpool hitrates are probably low. See SAP manual 51014418 "SAP on DB2 UDB for OS/390 and z/OS: Database Administration Guide" regarding buffer pool isolation, and evaluating size of bufferpools and hiperpools.
  - If average suspension time is bad, look for I/O contention with z/OS tools such as RMF Monitor III and RMF Monitor I.
  - Analyze frequently accessed tables that might be candidates for DB2 hardware compression. Many SAP application tables compress well. Hardware compression will store more rows per page, which generally helps increase hitrates and reduce I/O.
- High "Commit phase II" time (formerly captured under "service task switch"):
  - Not a frequent problem. This occurs on very large change intensive systems, or when the log datasets have been configured incorrectly.
  - Check performance of I/O to logs
  - Check configuration of logs – they should be configured with logs on different disks to minimize I/O contention between logging and archiving
  - Review implementing compression of tables with high change frequency, to reduce data written to log. (Compression can exacerbate a page latch contention problem, since each page contains more rows when the data is compressed.)
- High "Other read suspension" time:
  - Usually the second largest source of delay in DB2

  - Check for inefficient SQL – if the SQL predicates and table indexes are not well matched, DB2 often chooses an access path (table scan, hybrid join, etc) that will use prefetch. If the SQL problem is fixed, the inefficient access path is often replaced with an indexed access path, which references fewer pages, and does not have to use prefetch.
  - Check for I/O constraint using RMF III and RMF I
- High "Other write suspension" time:
  - This is not seen often. It occurs on change intensive batch workoads
  - Check disk write performance – check I/O contention, and I/O indicators, such as write activity and write cache misses with tools such as RMF I, RMF III, or ESS Expert
- High "Lock/latch suspension" time:
  - Is almost always logical (row) locking, which is fundamentally an application or data design issue.
  - Check for row lock contention on un-buffered number ranges, or number ranges that are buffered using only a small block of numbers. See section 9.2.7.
  - Find the tables causing the suspensions. With recent versions of SAP, this can be done with ST04 statement cache. Companies without RFCOSCOL ST04 may do this with lock suspension trace (IFCID 44,45), or by reviewing ST04 statement cache and looking for change SQL with long total elapsed time.
  - Find the programs causing the suspensions, using ST04 cache analysis, if necessary followed by SE11 "where used".
  - Investigate SAP settings that may help. As examples, we have seen locking problems with RSNAST00 resolved by program options, and locking problems in financial postings resolved by using "posting block" and making process changes. These changes are business process specific, and would need to be researched in OSS after the table and program with the locking problem are found.
  - Review possible application changes, such as grouping changes in SAP internal tables to be processed together just before commit, to reduce the time that locks are held in DB2.
  - Control level of parallelism of batch jobs and update processes, to maximize throughput.
  - (The above assumes that the system design is set, and cannot be changed to alleviate a locking constraint. System design would include issues such as the number of ledger entries and number of entries in statistics tables. Fewer statistics or ledger rows will lead to more lock contention, since more information is being aggregated into a few rows.)
- High "Page latch suspension" time:
  - This is not seen often. It occurs on very large change intensive systems.
  - Concurrent updates to a page by different threads will cause page latch contention on data pages.
  - High insert activity can also cause page latch contention on spacemap pages.
  - Run page latch suspension trace (IFCID 226,227) to confirm whether data pages or spacemap pages are causing suspension.
  - If the problem is not space map pages, but updates to different rows in same page, consider reducing MAXROWS on the table. This will increase I/O activity and reduce bufferpool hitrates, while reducing page latch contention.
  - If the problem is high activity on spacemap pages, consider partitioning to distribute the insert activity to different partitions or consider using the MEMBER CLUSTER option

on the table. MEMBER CLUSTER will reduce the number of pages mapped by each spacemap page. It will cause the clustering index to become disorganized faster.
    - Consider changing the clustering sequence on the table, to spread activity through the table. Verify that this will not cause performance problems for other programs that reference the table.
- High "Open/Close" time:
    - This occurs when the number of frequently accessed datasets in an SAP system is larger than maximum open datasets, which is controlled by the DB2 DSMAX parameter
    - Increase DSMAX, after evaluating the impact on DBM1 VSTOR using SAPnote 162293.
    - Confirm that the catalog for the datasets in the DB2 database is cached in VLF
- High "Global lock" time:
    - This occurs with DB2 data sharing, which is beyond the scope of this paper.
- High "Not attributed" time (displayed as "Other" in some SAP versions):
    - Check OS paging and CPU utilization on DB server – RMF I, II, and III
    - Check WLM priorities of DB2 and other address spaces, to confirm that the ICLI and DB2 have the correct priority.

8.2. DB2 delay analysis examples

8.2.1. Good ST04 global times

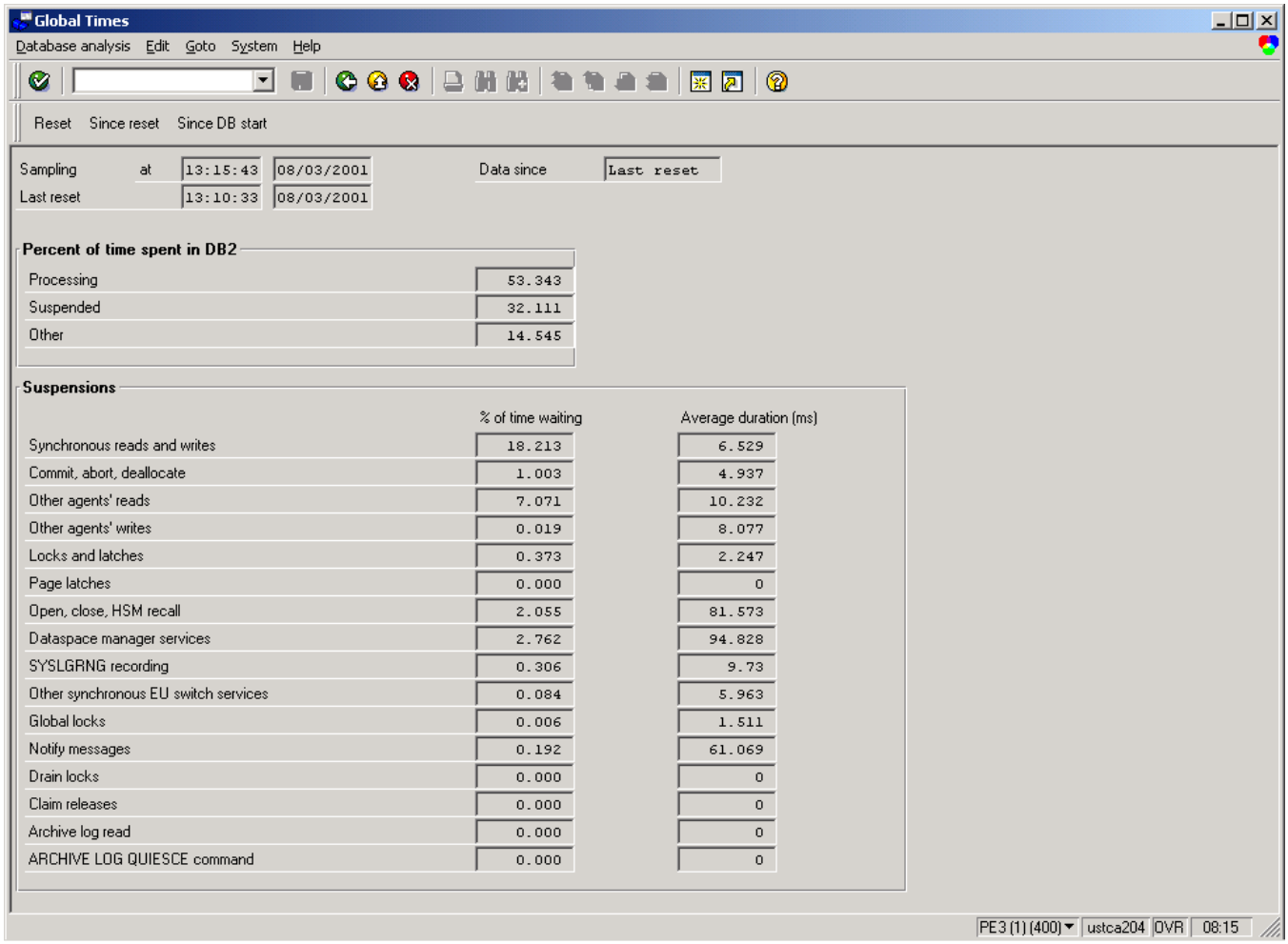


Figure 112: Good ST04 times

In Figure 112, the CPU time in DB2 (processing time) is over 50% of the time in DB2. This is our first filter for good DB2 performance. The average times for synchronous read and write are good – under 10 ms. "Other agent read" prefetch time is very good at 10 ms. As is often the case, "Synchronous read and write" (also reported as "I/O suspension" in some versions) is the largest component of delay.

The one indicator that is somewhat high is "Other" (Not attributed) at nearly 14%. This is often under 10%. If z/OS is usually running at high utilizations (80% and up), ST04 times will often show "other" or "not attributed" time of 10% to 20%. You should expect that "Not attributed" or "Other" time would run a bit higher if the DB server often runs at high CPU utilization.

8.2.2. Rather suspicious ST04 times

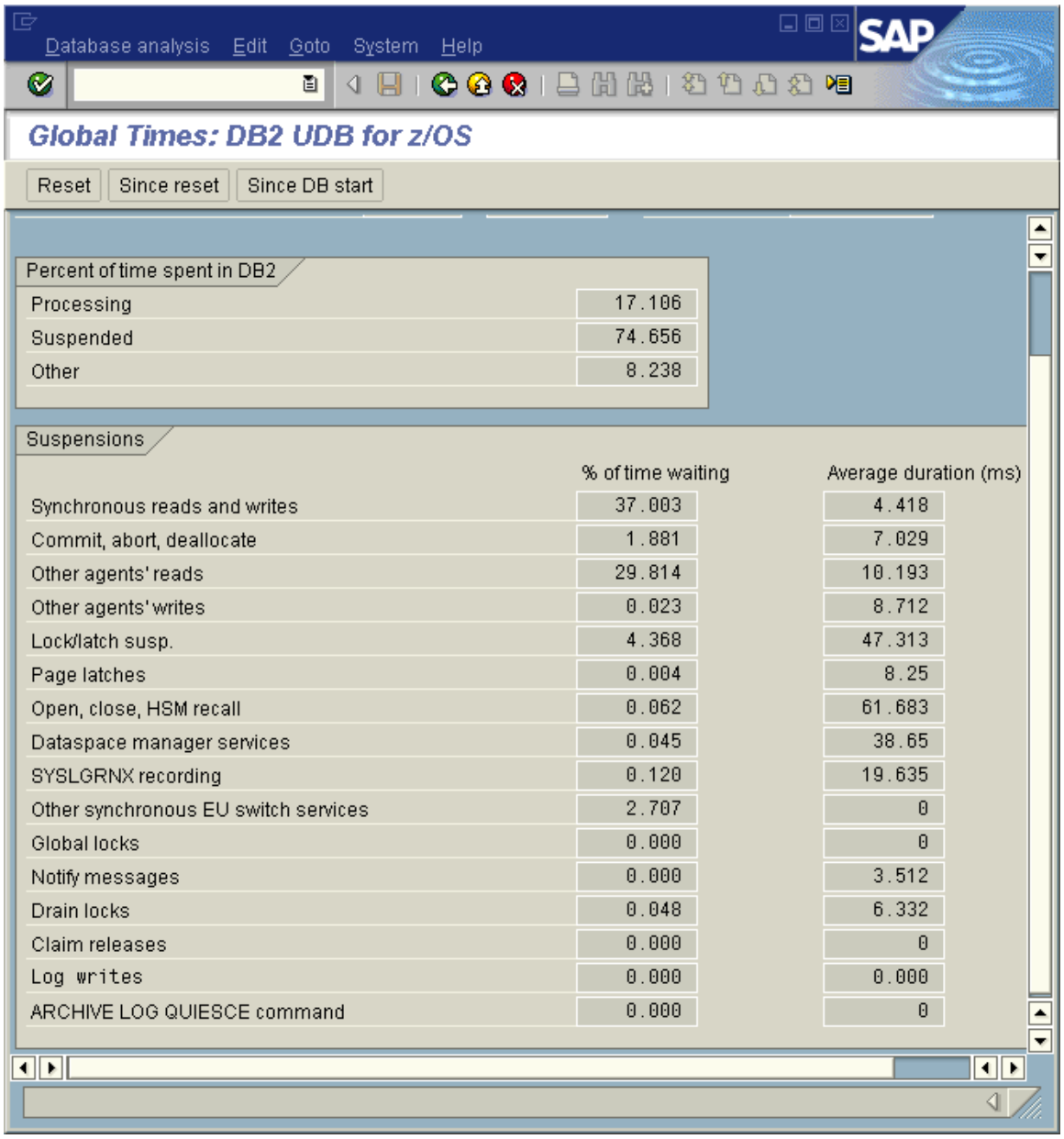
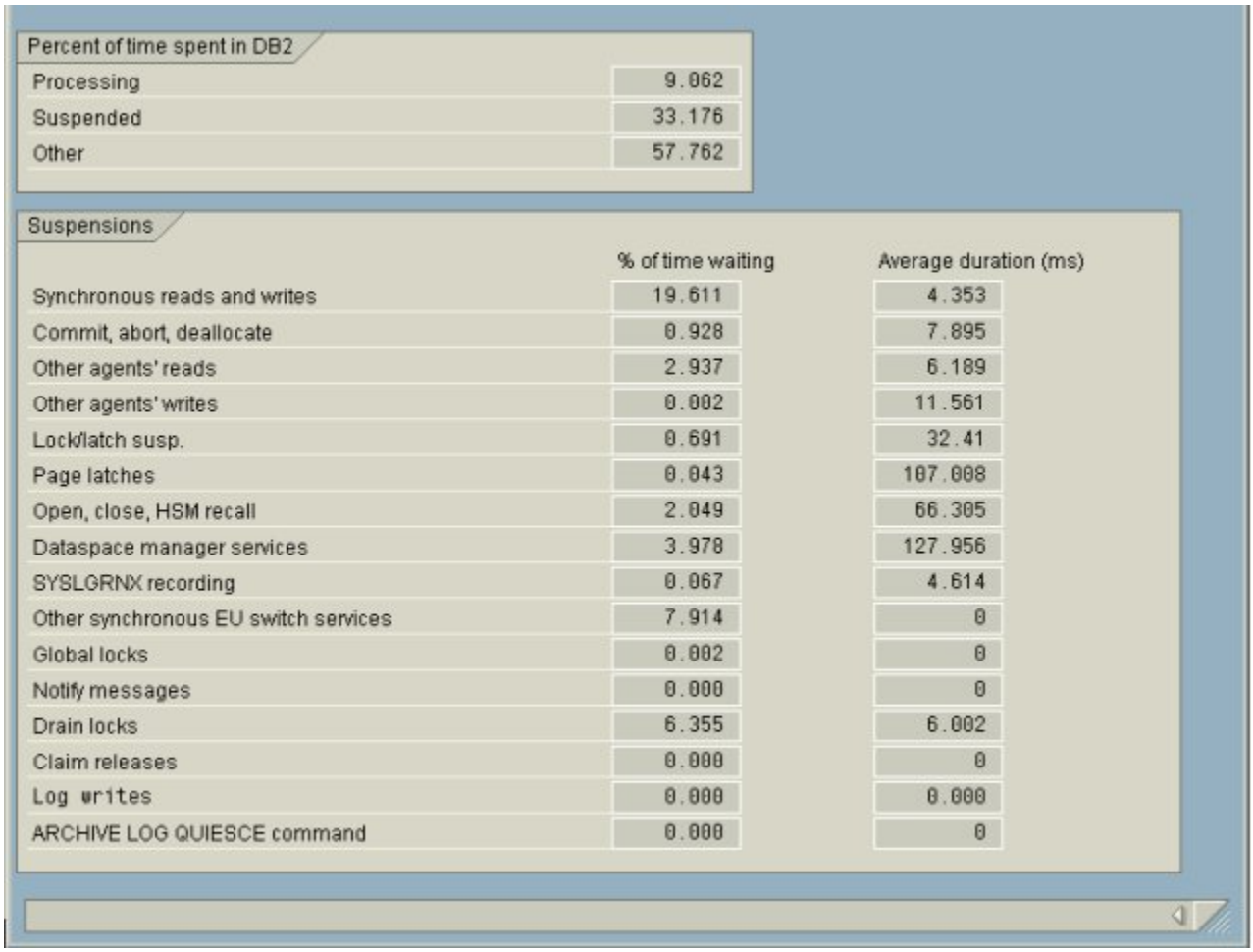


Figure 113: ST04 with long total suspensions

In Figure 113, the delay time is about 82% (74.6+8.2) with CPU about 17%, which is at the high end of normal delay in DB2. Average read times for synchronous read (4.4 ms) and other agent read (10 ms) are both ok, so the problem is likely a low hit rate, rather than slow I/O. We need to check the SQL cache, and look for inefficient SQL (Section 8.4).

8.2.3. ST04 Times points to constraint on DB server



Percent of time spent in DB2

Processing	9.062
Suspended	33.176
Other	57.762

Suspensions

	% of time waiting	Average duration (ms)
Synchronous reads and writes	19.611	4.353
Commit, abort, deallocate	0.928	7.895
Other agents' reads	2.937	6.189
Other agents' writes	0.002	11.561
Lock/latch susp.	0.691	32.41
Page latches	0.043	107.008
Open, close, HSM recall	2.049	66.305
Dataspace manager services	3.978	127.956
SYSLGRNX recording	0.067	4.614
Other synchronous EU switch services	7.914	0
Global locks	0.002	0
Notify messages	0.000	0
Drain locks	6.355	6.002
Claim releases	0.000	0
Log writes	0.000	0.000
ARCHIVE LOG QUIESCE command	0.000	0

Figure 114: ST04 times with high "Other in DB2"

ST04 times shows "Other" time is very high – 57%. This points to a problem on the database server -- usually a storage or CPU constraint. From SAP, we could use OS07 to get a snapshot of activity, to see if the problem is still occurring. (Since "thread times" is historical information, we may need to go back to performance history statistics, using RMF I, to check the problem)

Next actions would be to:

- Review historical CPU statistics in RMF I, to verify whether this occurs often
- Check SQL cache for inefficient SQL
- Evaluate operational changes such as limits on batch
- Review changing LPAR CPU weights to give the LPAR more CPU
- Etc.

8.3. Impact of data skew and parameter markers on prepare

The access path chosen by DB2 can in some cases be different when the statement is prepared with parameter markers (the original SAP R/3 way) or with parameter values (via ABAP hint described in SAP note 162034) or REOPT(ONCE) bind option.

When you are evaluating a statement, and want to check whether it will choose a different access path with values or markers, you can use a process such as the one outlined in this section. It is a way to test the impact of prepare with literals on a productive system, or other system, without having to modify the ABAP source code.

As a first step, trace the statement execution with ST05. This example is run on a system that is not using the REOPT(ONCE) BIND option.

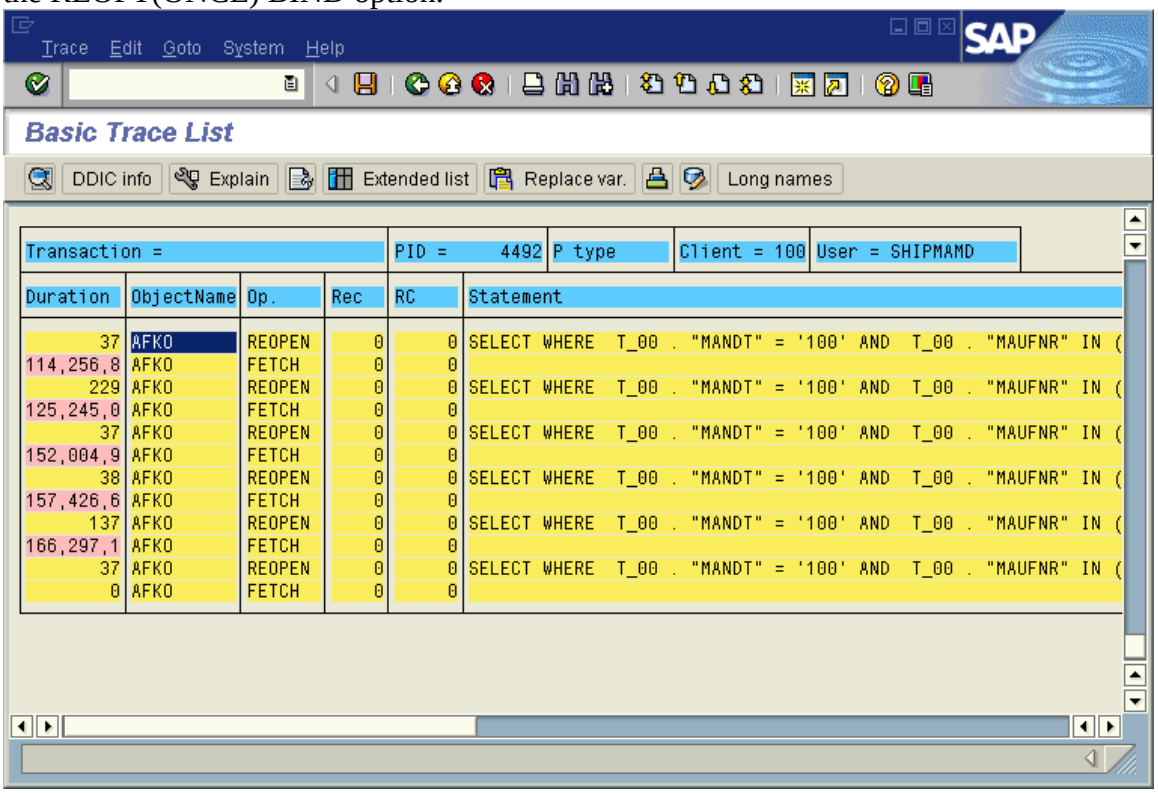


Figure 115: Slow join on AKFO

Performance is not good. Each call takes 120-150 seconds (Duration is in micro-seconds and if the number is too large, digits on the right are lost) and returns no rows (Rec is 0).

If you select the 'REPOPEN' line, and press explain, explain will be done with parameter markers. Statements prepared with parameter markers cannot make use of DB2 frequency distribution statistics. Prepare with markers is the normal way statements are prepared, without REOPT(ONCE) or an ABAP HINT.

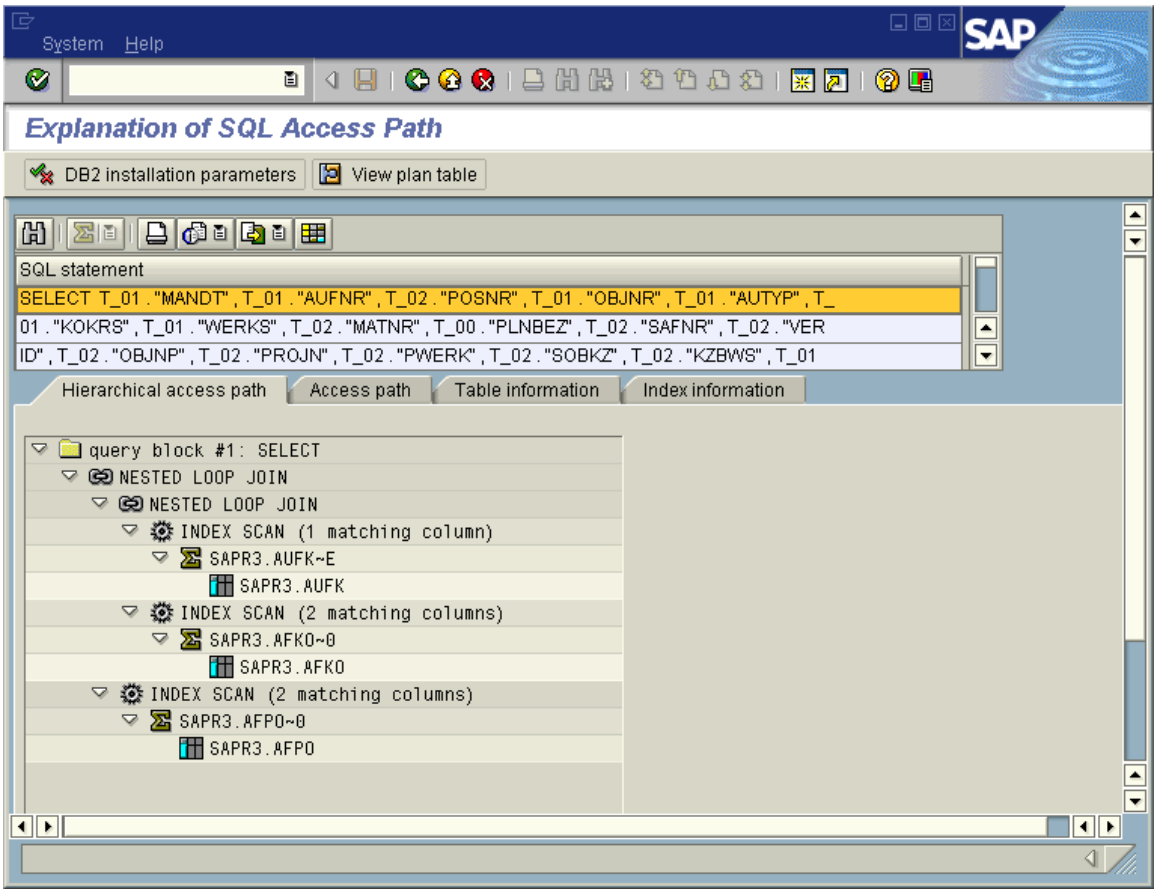
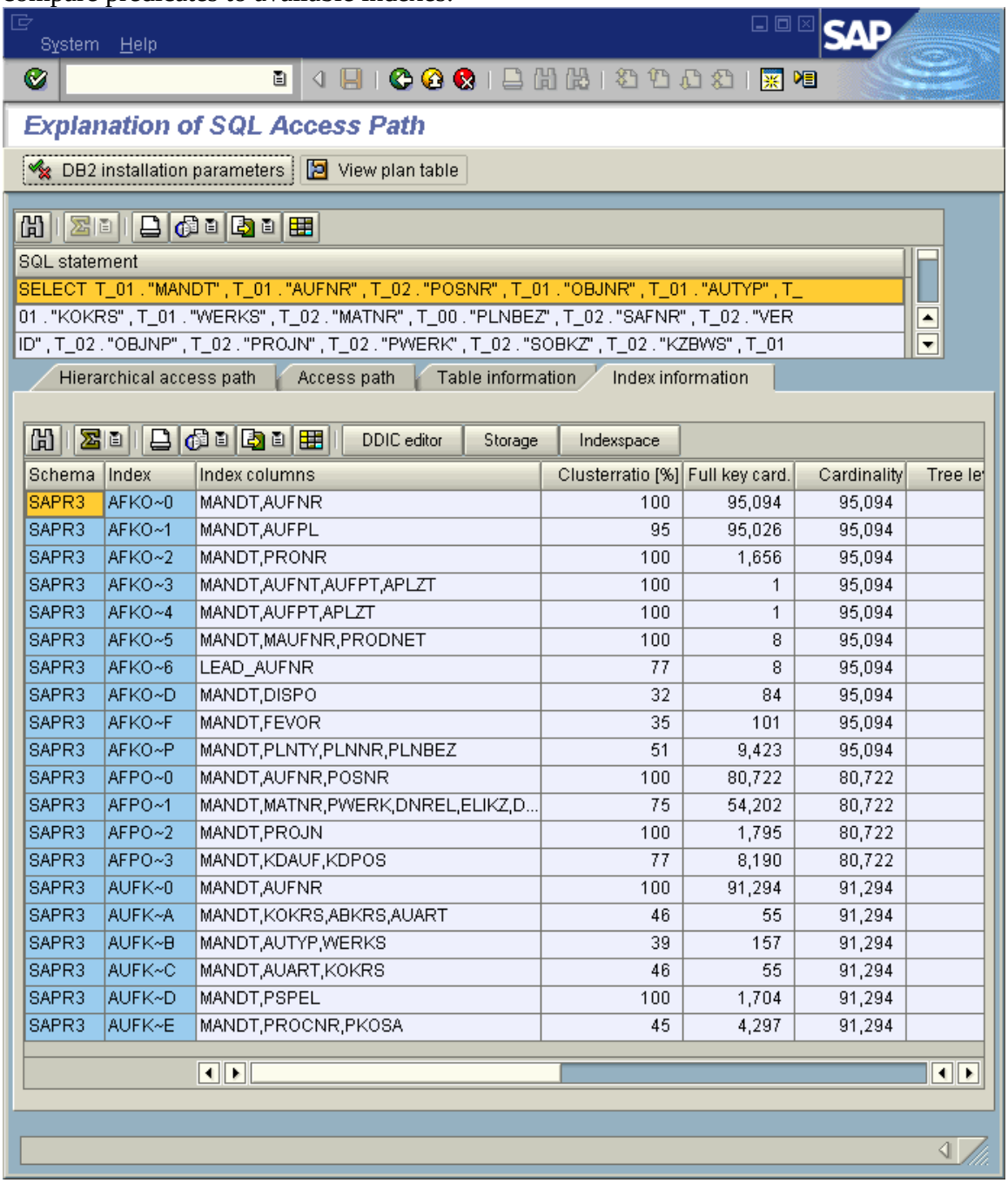


Figure 116: AFKO explained with parameter markers

The driving table of the join is AUFK, using index AUFK~E. The join order is AUFK, AFKO, AFPO. Since each call shown in ST05 took 120 to 150 seconds, and returned no rows, we know this was not a good choice. A three-way join that is well indexed should take just a few ms per row.

Select the “Index information” tab to display all indexes on all tables. This information will be used to compare predicates to available indexes.



Schema	Index	Index columns	Clusterratio [%]	Full key card.	Cardinality	Tree le
SAPR3	AFKO~0	MANDT,AUFNR	100	95,094	95,094	
SAPR3	AFKO~1	MANDT,AUFPL	95	95,026	95,094	
SAPR3	AFKO~2	MANDT,PRONR	100	1,656	95,094	
SAPR3	AFKO~3	MANDT,AUFNT,AUFPT,APLZT	100	1	95,094	
SAPR3	AFKO~4	MANDT,AUFPT,APLZT	100	1	95,094	
SAPR3	AFKO~5	MANDT,MAUFNR,PRODNET	100	8	95,094	
SAPR3	AFKO~6	LEAD_AUFNR	77	8	95,094	
SAPR3	AFKO~D	MANDT,DISPO	32	84	95,094	
SAPR3	AFKO~F	MANDT,FEVOR	35	101	95,094	
SAPR3	AFKO~P	MANDT,PLNTY,PLNNR,PLNBEZ	51	9,423	95,094	
SAPR3	AFPO~0	MANDT,AUFNR,POSNR	100	80,722	80,722	
SAPR3	AFPO~1	MANDT,MATNR,PWERK,DNREL,ELIKZ,D...	75	54,202	80,722	
SAPR3	AFPO~2	MANDT,PROJN	100	1,795	80,722	
SAPR3	AFPO~3	MANDT,KDAUF,KDPOS	77	8,190	80,722	
SAPR3	AUFK~0	MANDT,AUFNR	100	91,294	91,294	
SAPR3	AUFK~A	MANDT,KOKRS,ABKRS,AUART	46	55	91,294	
SAPR3	AUFK~B	MANDT,AUTYP,WERKS	39	157	91,294	
SAPR3	AUFK~C	MANDT,AUART,KOKRS	46	55	91,294	
SAPR3	AUFK~D	MANDT,PSPEL	100	1,704	91,294	
SAPR3	AUFK~E	MANDT,PROCNR,PKOSA	45	4,297	91,294	

Figure 117: AFKO join - index display

From the SQL trace in Figure 115, use the 'replace var' button to view the statement with all parameter values filled in.

The local predicates are MANDT (on all three tables), AFKO.MAUFNR, AUFK.LOEKZ, and AUFK.PKOSA. If any of these has an uneven distribution of values, then prepare with literals may allow DB2 to choose a different, and better, access path.

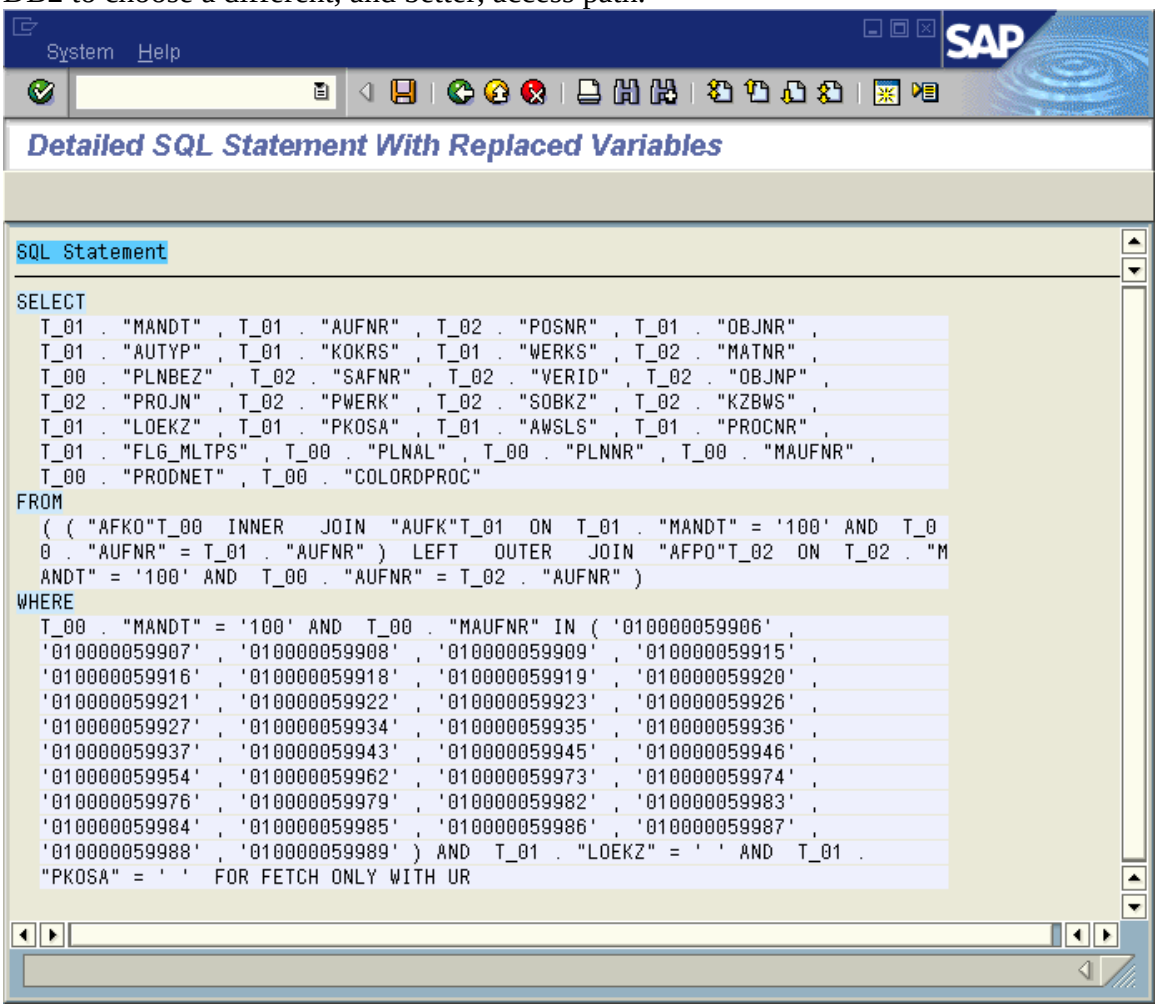


Figure 118: AFKO statement with replaced variables

Copy the statement to the windows clipboard with the GUI cut and paste, or download to a file using the %PC OK code or the "system > list > download" menu path. We will use this later to explain the statement with values.

Comparing the local predicates with the indexes, we see that

- AUFK~E (MANDT, PROCNR, PKOSA)
- AFKO~5 (MANDT, MAUFNR, PRODNET)

Both support evaluating local predicates on an index. AUFK.LOEKZ is not in any index, so DB2 must read the table row to determine whether the row satisfies the local predicate LOEKZ=' '.

As shown earlier, in the Table Information tab, one can select a table and display column cardinalities.

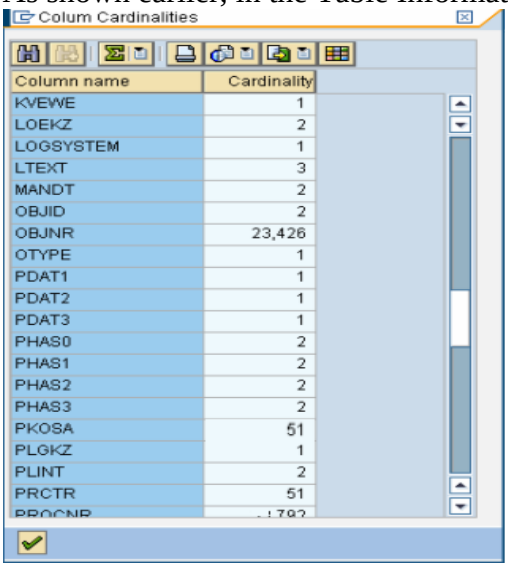


Figure 119: AUFK Column Cardinalities

The cardinalities for this example are (not all are displayed here)

- PKOSA – 51
- LOEKZ – 2
- MAUFNR – 8

DB2 then estimates how many rows will be retrieved using the ratio of the cardinality of the column and the number of rows in the table. For instance, if there are 51 values of PKOSA and 90,000 rows in AUFK, then on the average "PKOSA =" will retrieve 90,000/51 rows, that is 1764 rows. Likewise MAUFNR IN is estimated to retrieve 8/90000 rows for each IN list element. So in this case, DB2 considers that MAUFNR IN will retrieve many values – that it is not filtering.

But if the values are not distributed uniformly in all rows, then it could be that PKOSA=xxx retrieves many more values than PKOSA=yyy, or that the number of rows matching MAUFNR IN is far fewer than the estimate of 8/90000.

In a case where we know that the SQL is being executed inefficiently, we need to check whether there might be skew (non-uniform distribution) that is causing DB2 to estimate incorrectly and choose the wrong access path. (In this example, it could also be the case that the filtering local predicate is on LOEKZ, which is not indexed.

To find out if there is skew, use a query "select column, count(*) from table group by column order by 2 desc;".

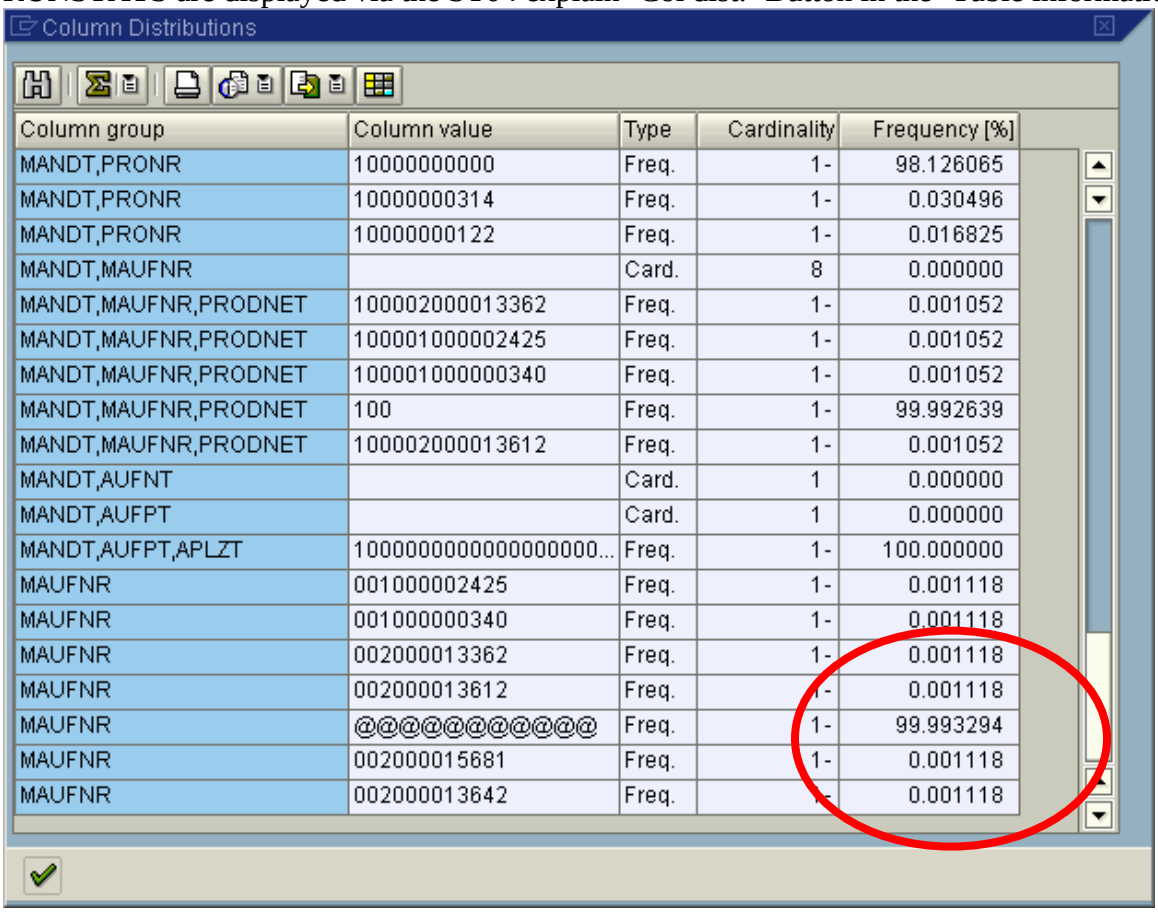
```
SELECT MAUFNR, COUNT(*) FROM AFKO
GROUP BY MAUFNR
ORDER BY 2 DESC ;
```

	MAUFNR	
1_		96307
2_	014200204352	146
3_	014000136735	81
4_	014000187506	80
5_	014000025390	79
6_	014000153405	77
7_	014000087950	77
8_	014000110384	76
9_	014000129038	75

Figure 120: Query to check for skew

In Figure 120, we see that there is not a uniform distribution of rows in all values, and that “spaces” is the value that occurs in almost all rows. Since our SQL in Figure 118 includes numbers for MAUFNR, we know that if the number is found, very few rows are returned. So, MAUFNR IN is a filtering local predicate when the value is not spaces.

Perform RUNSTATS on the tables using FREQVAL on the MAUFNR column, to gather column distribution information. See SAP note 1008334 for examples of the syntax. Here, the statistics after the RUNSTATS are displayed via the ST04 explain 'Col dist.' Button in the 'Table information' tab.



Column group	Column value	Type	Cardinality	Frequency [%]
MANDT,PRONR	10000000000	Freq.	1-	98.126065
MANDT,PRONR	10000000314	Freq.	1-	0.030496
MANDT,PRONR	10000000122	Freq.	1-	0.016825
MANDT,MAUFNR		Card.	8	0.000000
MANDT,MAUFNR,PRODNET	100002000013362	Freq.	1-	0.001052
MANDT,MAUFNR,PRODNET	100001000002425	Freq.	1-	0.001052
MANDT,MAUFNR,PRODNET	100001000000340	Freq.	1-	0.001052
MANDT,MAUFNR,PRODNET	100	Freq.	1-	99.992639
MANDT,MAUFNR,PRODNET	100002000013612	Freq.	1-	0.001052
MANDT,AUFNT		Card.	1	0.000000
MANDT,AUFPT		Card.	1	0.000000
MANDT,AUFPT,APLZT	1000000000000000000...	Freq.	1-	100.000000
MAUFNR	001000002425	Freq.	1-	0.001118
MAUFNR	001000000340	Freq.	1-	0.001118
MAUFNR	002000013362	Freq.	1-	0.001118
MAUFNR	002000013612	Freq.	1-	0.001118
MAUFNR	@@@@@@@@@@@@	Freq.	1-	99.993294
MAUFNR	002000015681	Freq.	1-	0.001118
MAUFNR	002000013642	Freq.	1-	0.001118

Figure 121: SKEWED data distribution in MAUFNR column

Note the uneven distribution in the values of MAUFNR – in 99% of the cases it is not a number, so the local predicate MAUFNR IN (number1, number2, number3) should help to eliminate extraneous rows, since it is searching for rows with numbers.

Next, we explain the statement with literals. Either use the ST05 "Explain one statement" button, or from an SQL trace as shown in Figure 115, press the "Editor" button (second from right). Paste or upload the saved statement with parameters. You may need to edit the text to quote literals, or remove trailing blanks from lines.

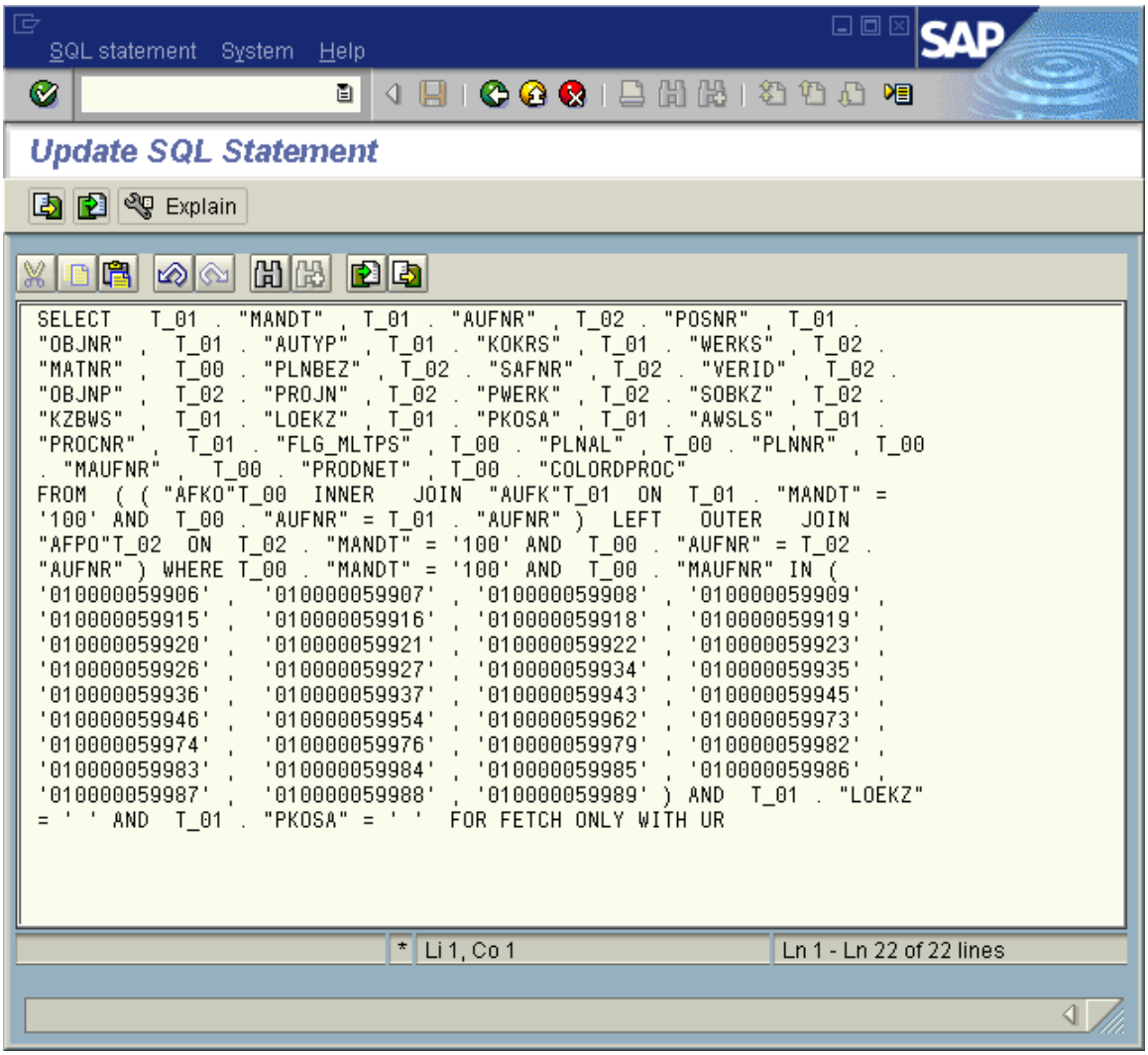


Figure 122: AFKO statement with variables

Pressing 'Explain' will now explain this statement with values. This prepare with values allows DB2 to use column distribution information to prepare the statement.

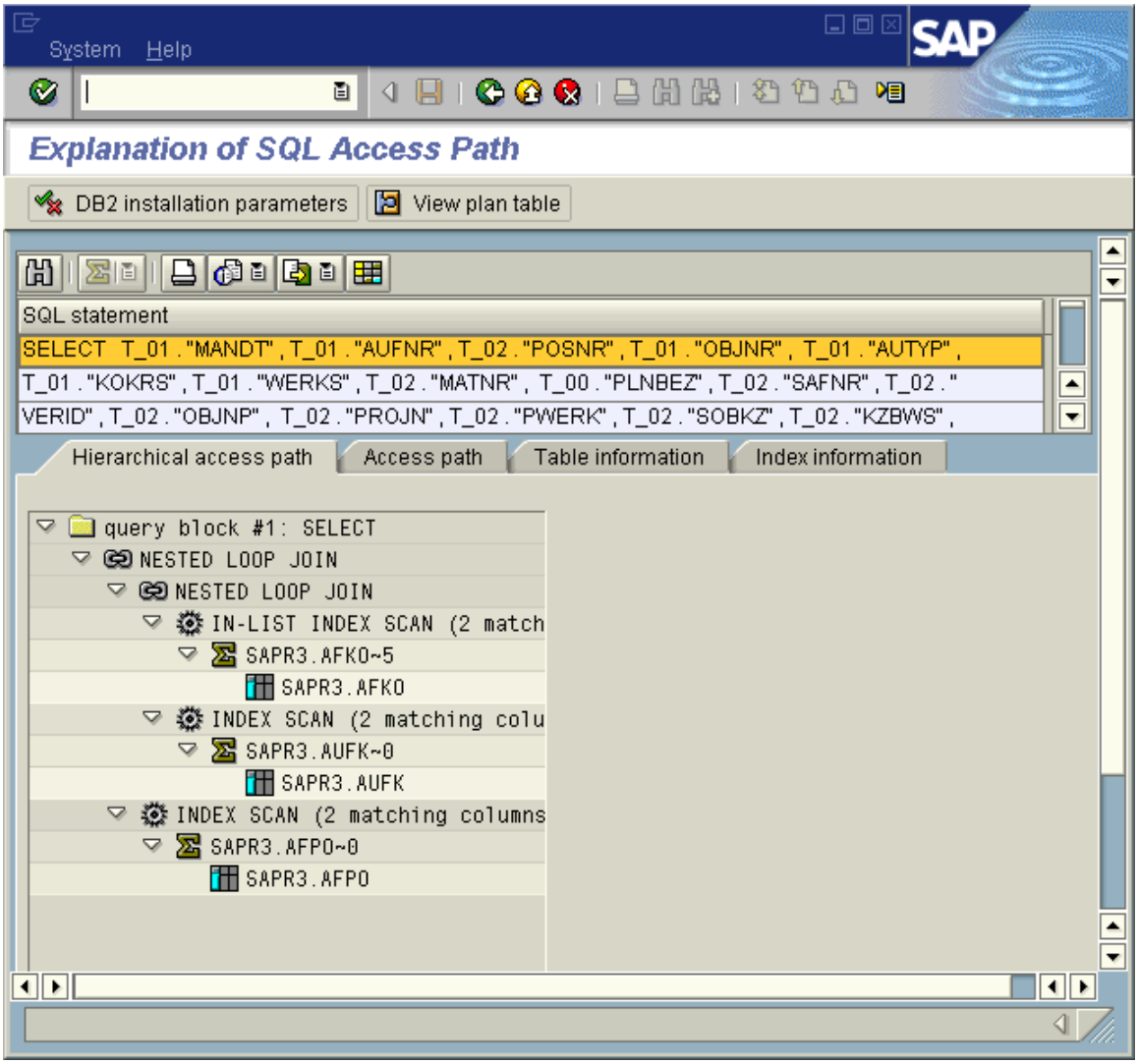


Figure 123: AFKO explain with variables chooses different access path

Now, the driving table is AFKO using index AFKO~5, which is MANDT, MAUFNR, PRODNET. The join order is AFKO, AUFK, AFPO.

DB2 used the frequency distribution for MAUFNR (shown in Figure 121) to determine that this index would be a good choice. The FREQVAL statistics together with the statement values tell DB2 that the statement is looking for values that seldom occur in MAUFNR, so this index with the new join order will eliminate many candidate rows when the first table in the join is accessed.

In addition to the count SQL in Figure 120, one can also use SE16, with the parameter values, to determine whether the predicates on AFKO filter well. Here, run SE16, and fill in the values taken from the SQL 'replace vars' above.

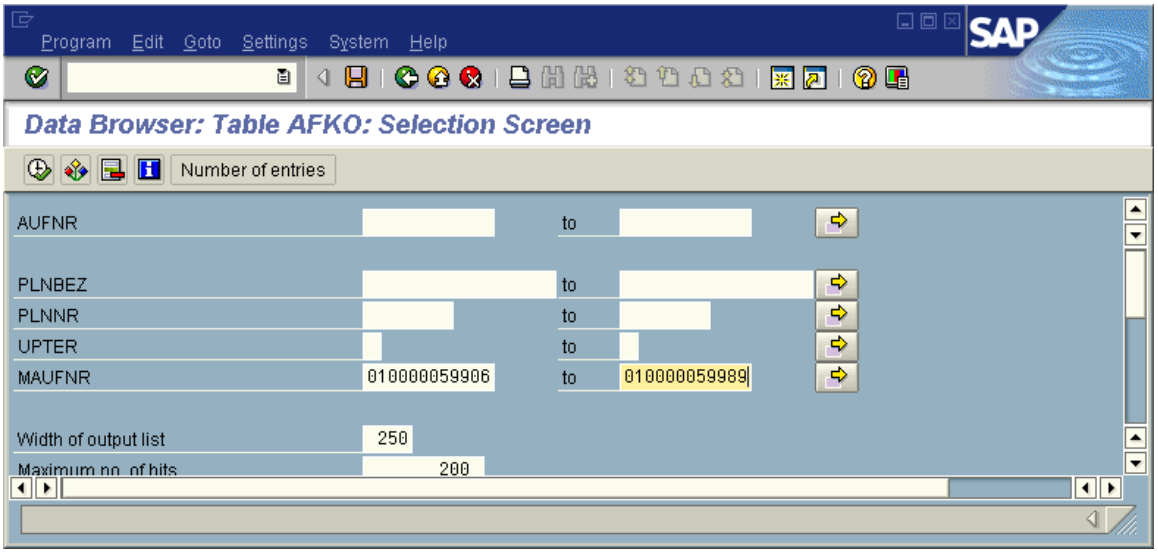


Figure 124: SE16 to test predicate filtering

Press ‘number of entries’.

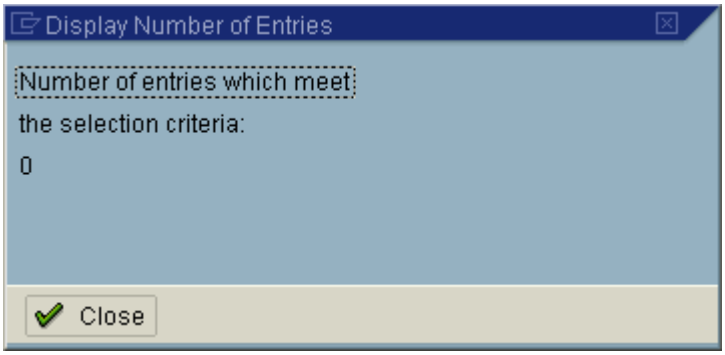


Figure 125: SE16 filtering test number of entries

The SE16 test shows that for this sample, the MANDT and MAUFNR column alone will eliminate all rows. (MANDT is implicitly used by SE16). Thus, if DB2 chooses the join order shown in Figure 123, it will be more efficient than the current access path.

At this point, we know that the solution requires that we make DB2 aware of the skew:

- We need to modify the ABAP source (SAP note 162034), to add a hint so that the statement will be re-optimized with values at execution time, and
- RUNSTATS with FREQVAL should be done on the AFKO table, and
- If DB13 is used for RUNSTATS, exclude AFKO from DB13 runstats.

8.3.1. REOPT(ONCE) as alternative to ABAP HINT

As an alternative to using ABAP HINTS, with DB2 V8 there is now a new bind option REOPT(ONCE) that can also help DB2 to prepare statements with values. With REOPT(ONCE), the first time a statement is prepared, it is prepared using the values, but the cached version of the statement has markers, so it can be executed for different input values. If there are DB2 frequency statistics on columns with skew, REOPT(ONCE) allows DB2 to determine the skew while still sharing the statement. REOPT(ONCE) affects the entire system. An ABAP HINT affects only a single program.

An ABAP HINT can be used to fix an individual program, but when the statement is prepared, the parameter values will be in the cached statement, so the statement cannot be shared for executions with different parameter values.

See SAP note 1008334 for more information on REOPT(ONCE).

If the problem in section 8.3 were to be solved using REOPT(ONCE), the solution would be:

- Gather FREQVAL statistics on MAUFNR in AFKO table
- Bind the DB2 plans for SAP with REOPT(ONCE) (which affects the entire system)
- Do not add HINT to ABAP.

8.4. Process for identifying slow or inefficient SQL

When starting performance analysis from the DB server, the first step is to check the efficiency of the SQL issued by the SAP programs. That is, check that the SQL matches the available indexes, and so does not have to search too many data and index pages to return a result. Many DB and OS performance problems (low bufferpool hit rates, high CPU utilization, I/O bottlenecks, etc) can by symptoms of inefficient SQL. Before trying to alleviate problems in these areas, it is best to check whether the root cause is inefficient SQL.

The SAP DBACOCKPIT (ST04) transaction is used to examine the DB2 statement cache, in order to review the SQL that is currently executing, or was recently executed, on the DB server. In a DB2 datasharing environment, this statement cache is specific to each active datasharing member, and can be separately reviewed on each DB2 subsystem, or can be merged in ST04 by selecting ALL for the DB2 datasharing member.

By default, statement performance statistics are not accumulated in DB2. In order to enable statement counters, the command “START TRACE(P) CLASS(30) IFCID(318) DEST(SMF)” can be used. Any of the user classes (30, 31, 32) can be specified. This does not actually write data to SMF, but enables gathering statement statistics in memory in DB2. Enabling IFCID 318 uses a small amount of CPU, but without it, it is nearly impossible do to performance analysis on an SAP system on DB2 for z/OS. If you are not doing performance analysis, and need to conserve CPU, IFCID 318 can be turned off.

The statement cache counters are accumulated for each statements in the statement cache. Statements that are not executed for a while can be pushed out of cache, at which time their statistics are lost. Statements that are executed relatively frequently can stay in cache for days or weeks. This means that if IFCIF 318 is always running, we don’t have a known starting point for statement statistics, and without a known starting

point it is difficult to compare the impact of different statements. It is helpful to stop and re-start IFCID 318 periodically when doing statement cache analysis. This does not affect the statements in the cache, but it resets all the statement counters. For instance, one could stop and start IFCID 318 in the morning, and then view the statistics during the day, to examine SQL that is run during the day.

This section describes and has examples of the indicators of inefficient SQL. Possible solutions to inefficient SQL are presented in a later section.

The key indicators of inefficient SQL are:

- High rows examined and low rows processed – this is a sign that DB2 is evaluating local predicates on the table rows, and not in the indexes.
- High getpages and low rows processed – this usually is a sign that DB2 cannot do matching index access, as when the local predicate columns are not all in the leading columns of the index, and DB2 must do index screening. It can also be caused by poor clustering, indirect references, or data fragmentation.
- Long statement average elapsed time – this can be a symptom of I/O constraints, row lock contention, or other delays.

The elapsed times of statements in ST04 statement cache do not include network time. It is time where the SQL is being executed on the DB server. This is different from SAP "database request time", which contains DB server time, as well as time communicating with the DB server over the network.

When searching the statement cache for inefficient SQL, it is helpful to sort the statement entries by total getpages, total rows processed, total elapsed time, or total CPU time and then use the three key indicators above (high rows examined/getpages and low rows processed, long average elapsed time) to find individual problem statements. The sort presents the high impact statements on the top of the list.

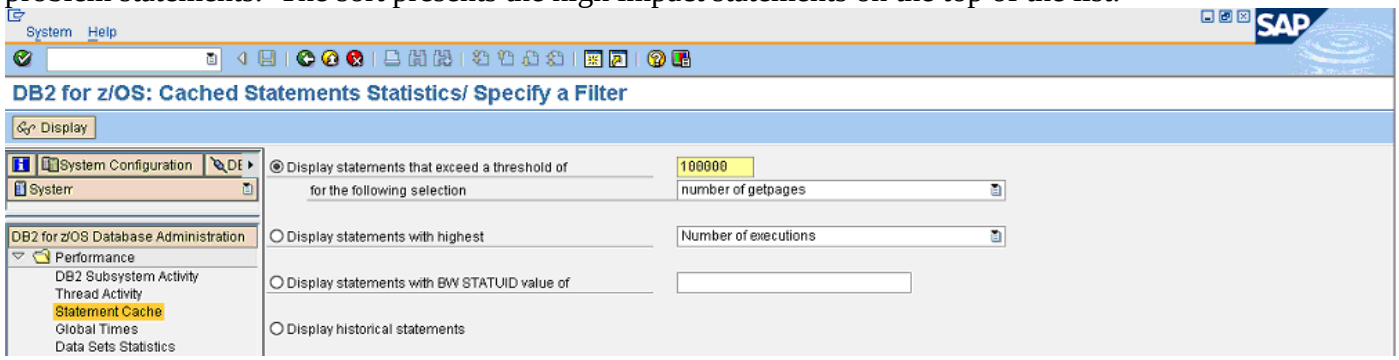


Figure 126: DBACOCKPIT to display statement cache

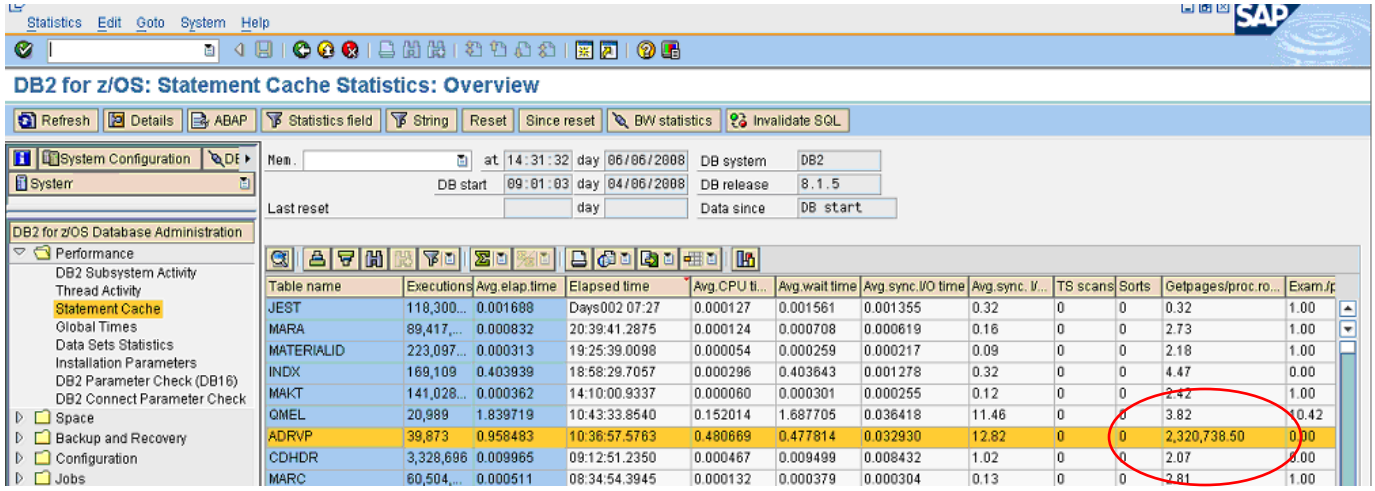


Figure 127: ST04 cached statement statistics sorted by elapsed time

In the example in Figure 127, the highlighted statement on ADRVP has high “getpages/processed row”, which is a sign of inefficient processing.

Select a statement and press “Details” to see the statement, as well as execution statistics.

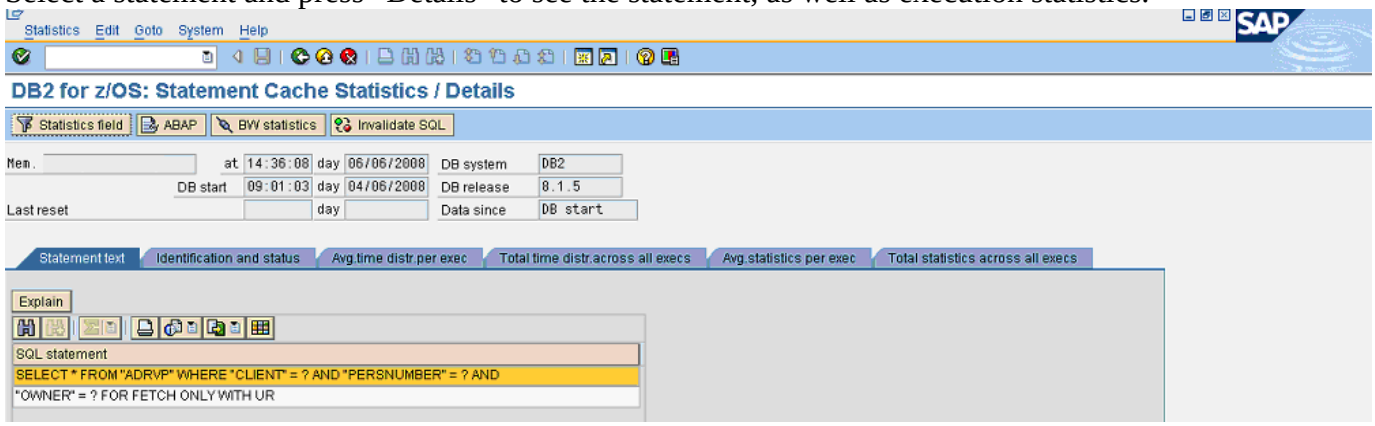


Figure 128: ST04 details - statement text

The statement text displays the local predicates (where column = value clauses) that can be matched to the indexes, to determine why the ratio of “getpages/processed rows” is high.

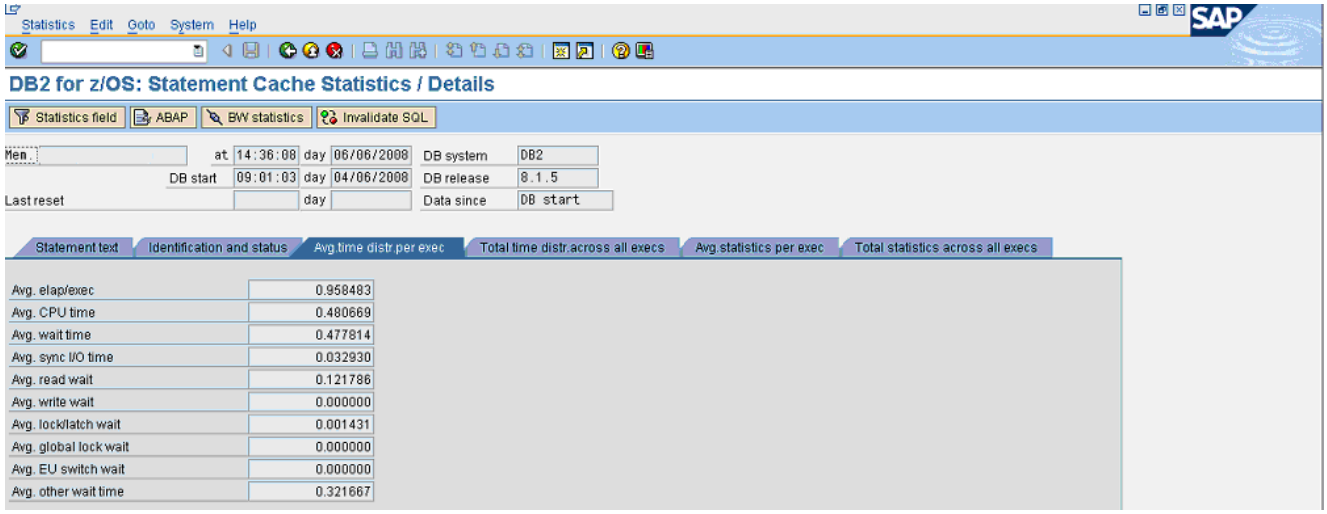


Figure 129: ST04 details - time per execution

Figure 129 shows that each execution runs nearly one second, and uses about ½ second of CPU.

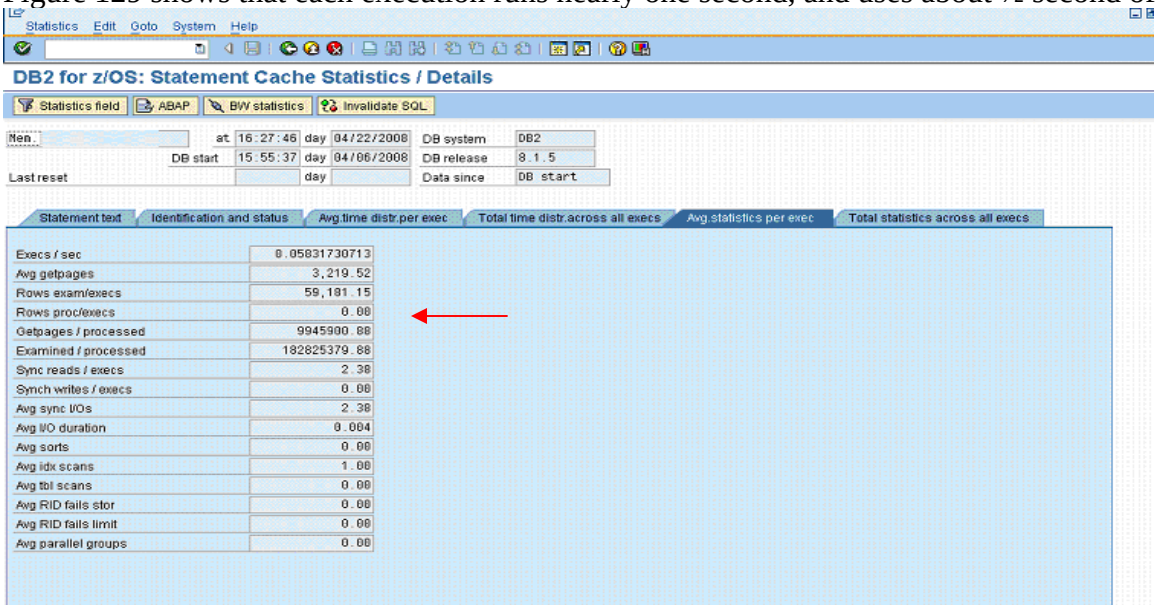


Figure 130: ST04 details – statistics per execution

But Figure 130 shows that each execution on the average returns no rows (Rows proc/execs). In general, a statement that returns no rows and which can effectively use indexes would use a fraction of one ms of CPU, and have an elapsed less than one ms, so the actual one second elapsed time and ½ second CPU per execution are high.

High rows examined per row processed in Figure 130 means that local predicates are being evaluated on table rows.

From the “statement text” tab in Figure 128, one can press explain to view the execution plan.

If it is available, explain will retrieve the actual access path in use in DB2. When you see the "Cached access path" is checked (see Figure 132), you know you are looking at the real access path used by DB2.

If cached access path flag is not checked, it is possible that explain plan for explain with parameter markers is not showing the actual access path that was used by DB2. One example of when this can occur is when REOPT(ONCE) is the bind option – in this case, DB2 will prepare the statement with parameter values, but the statement cache will not have the values. If there is skew in the data, then when the statement is prepared with markers from the cache the access path will be different.

If the cached access path is not available, you will see a popup:

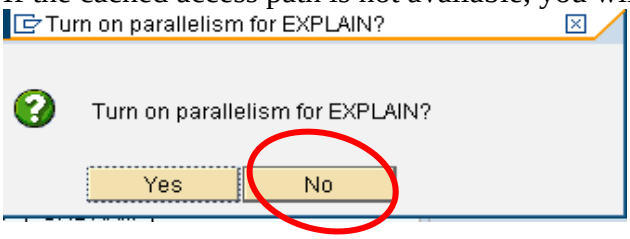


Figure 131: Turn on parallelism popup

Press "No", except if explaining query SQL on BI InfoProviders (Cubes, ODSes, DSOs, etc).

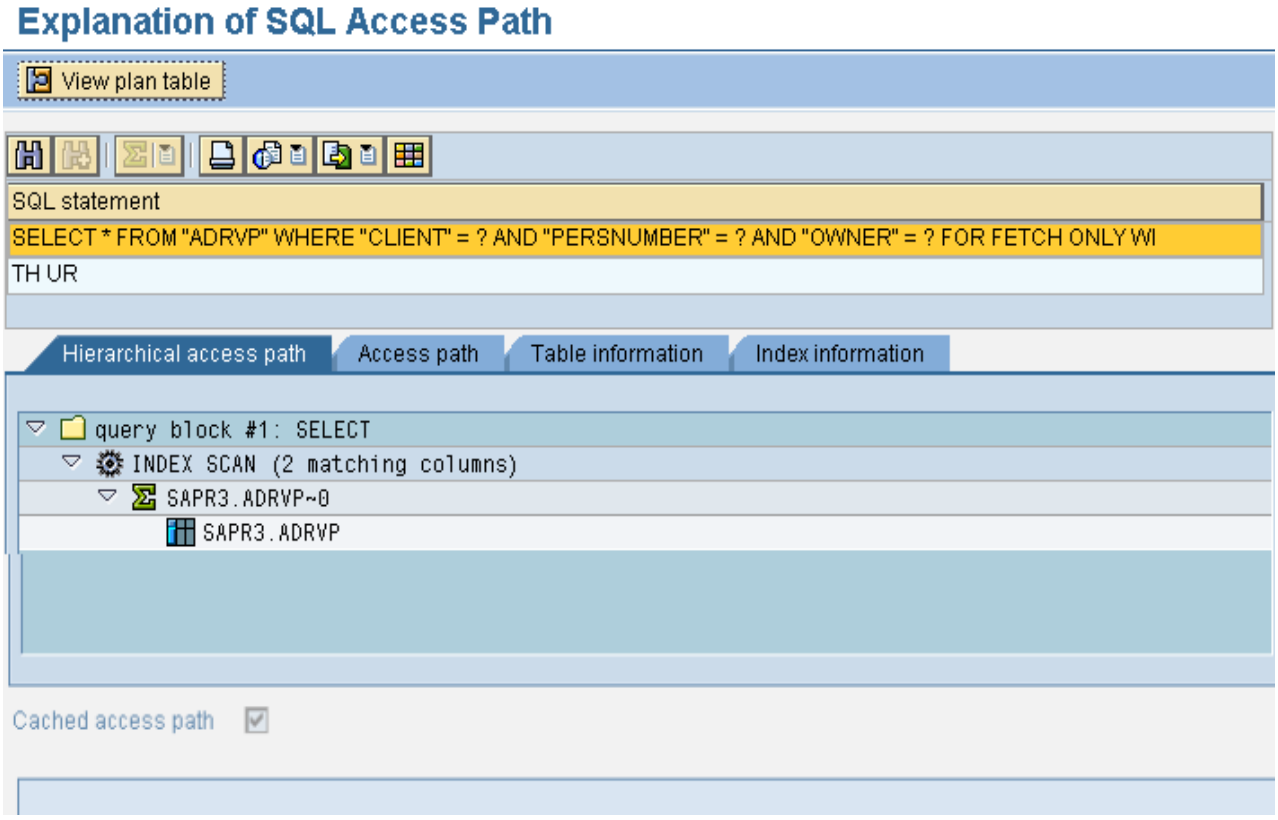


Figure 132: Explain Hierarchical Access Path

If you press the "Index Information" tab, the indexes are displayed, and can be compared with the local predicates.

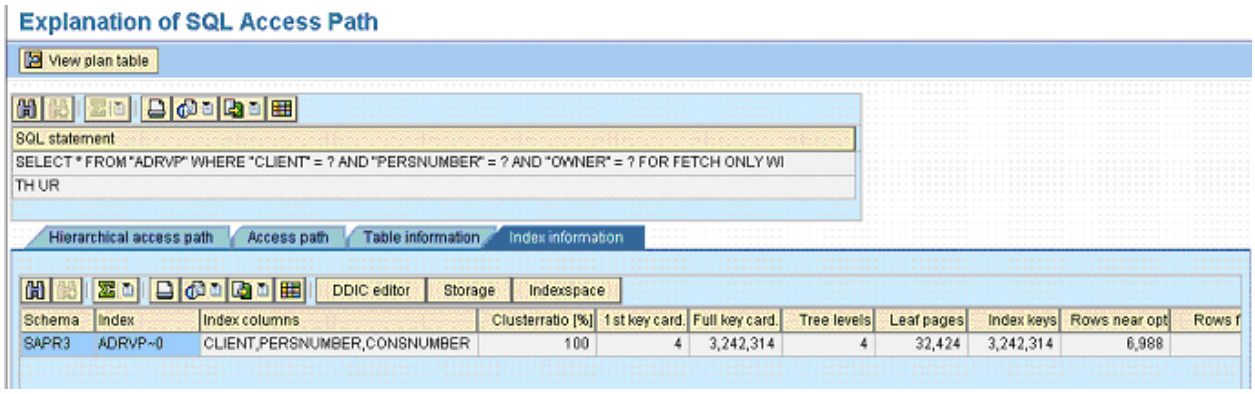


Figure 133: Explain Index Information

The local predicates in the statement are CLIENT=, PERSNUMBER=, and OWNER=. But the index only supports CLIENT= and PERSNUMBER=. OWNER is not in an index. In order to improve the performance of this statement, a new index on ADRVP would be needed. Since this is SAP code and an SAP table, one would search for SAP notes, and if none were found, open a message to SAP.

8.4.1. High getpages and low rows processed (per execution)

A getpage is when DB2 references a table or index page, in order to check the contents of the page. Examining many pages will use additional CPU, and contribute to pressure on the buffer pools. The situations where high "getpages per row processed" indicator will be seen are:

- Predicates contain columns which are not indexed, or not in the index used to access the table
- Predicates contain range predicates, which causes columns in the index to the right of the column with the range predicate to be evaluated with index screening rather than index matching
- Index screening, when there is an index column with no local predicate, but index columns on its lef and right with local predicates on the columns

In cases where a statement never returns a result, "Getpages/processed" in "Avg statistics per exec" is reported in ST04 as 0, since the quantity (getpages / 0) is undefined. If you see a high impact statement (high total rows, high total getpages, or long elapsed time) where "Getpages/processed" is 0, check the "Avg. statistics per execution" tab, and look at "Avg. getpages", which is a per-execution counter.

8.4.2. High rows examined and low rows processed (per execution)

A row is examined when DB2 checks a row in a table to determine if a row satisfies the predicates, or to return a row. If a row can be disqualified based on predicates that can be processed via index access, this does not count as a "row examined". When DB2 must read the rows in a table (rather than just the index) to determine whether a row satisfies the predicates, then "rows examined per row processed" can be high. Situations where this commonly happens are when:

- Predicates contain columns which are not indexed, or not in the index used to access the table

After finding a statement with high "rows examined per row processed", one should also check the average number of rows examined and rows processed, to confirm that the statement is inefficient.

In cases where a statement never returns a result, "Examined/processed" in "Avg statistics per exec" is reported in ST04 as 0, since the quantity (rows examined / 0) is undefined. If you see a high impact statement (high total rows, getpages, or long elapsed time) where "Examined/processed" is 0, check the "execution statistics", and look at "Avg. rows examined", which is a per-execution counter.

When index-only access is used, rows examined per execution will be 0, so "Examined/processed" will also be 0, regardless of how many rows are returned. Check "Getpages/processed" to evaluate if the statement is executed efficiently.

8.4.3. Long "average elapsed time" per execution

There are a number of reasons why one execution of a statement may take a long time. In the case where the statement fetches, inserts, or changes hundreds or thousands of rows, it is normal. Check "Avg rows processed" to see the number of rows returned per execution. In the situation where only few rows are processed on each execution, or the time per row processed is long, it could point to one of several things:

- I/O constraint on disk where the table or index resides
- Logical row lock contention, which is seen with change SQL and "select for update".
- Inefficient SQL as described above
- DB2 contention on page latches
- Indirect references or disorganized data cause excessive I/O

The Statement Cache "Avg. time dist per exec" tab will point out what the likely problem is, since it separates lock, I/O, and other waits into separate categories.

Figure 134 shows the statement cache sorted by total elapsed time. Individual statements can be reviewed, to find those that have long average elapsed time (Avg. elap. time).

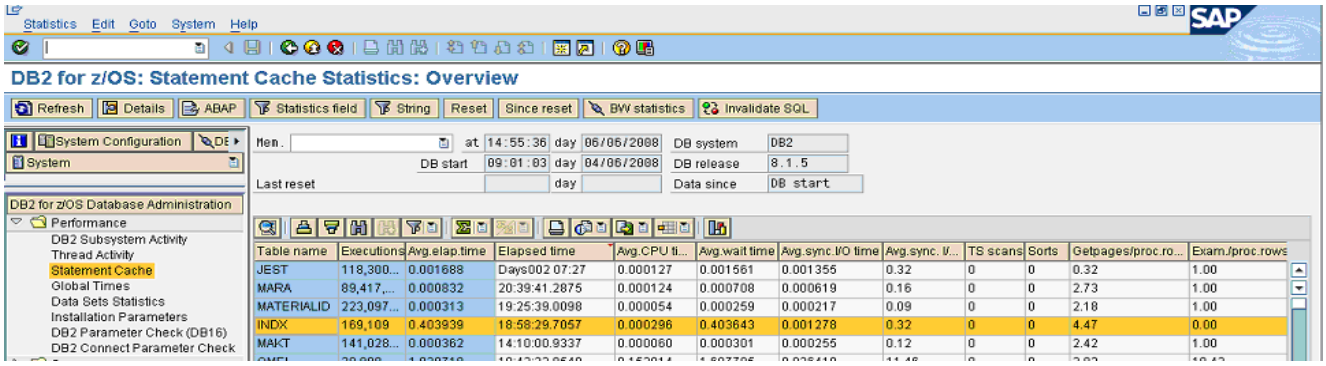


Figure 134: ST04 statement cache slow statement

Each statement takes nearly half a second (0.4039), and most of the time is wait (0.4036).

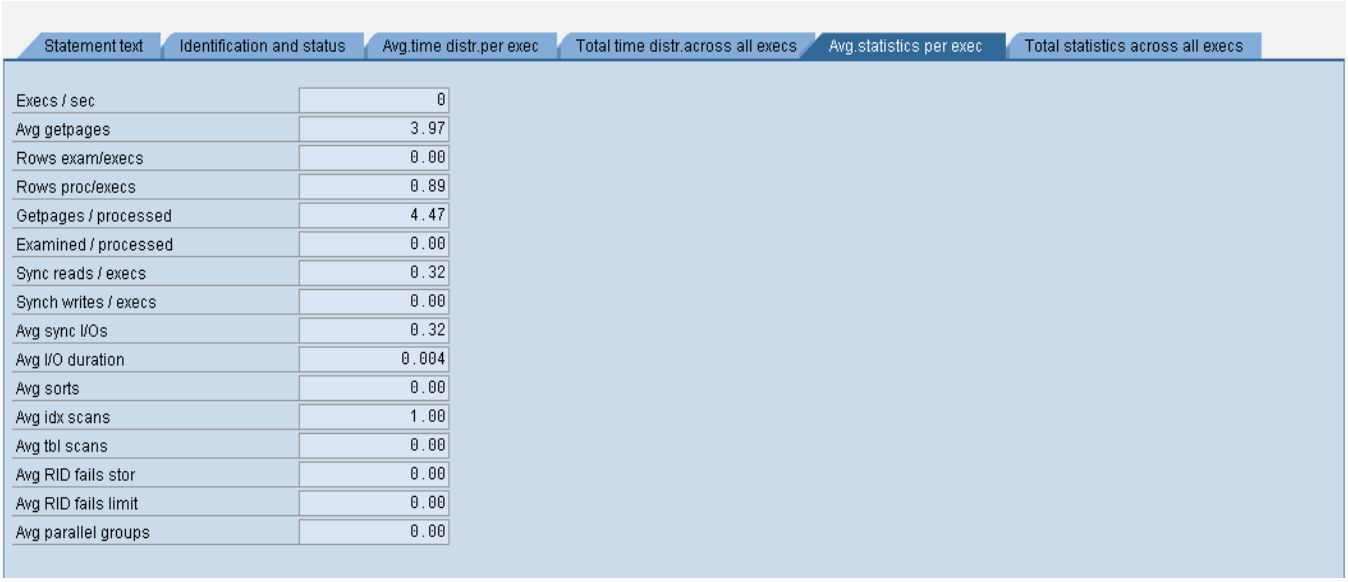


Figure 135: Slow SQL with few getpages

Figure 135 shows few getpages per execution (3.97) and little I/O per execution (0.32), so the long sql is not caused by processing lots of data, we can check time distribution.

Figure 136: ST04 time distribution

Figure 136 shows It is lock and latch wait, which is almost always an application issue.

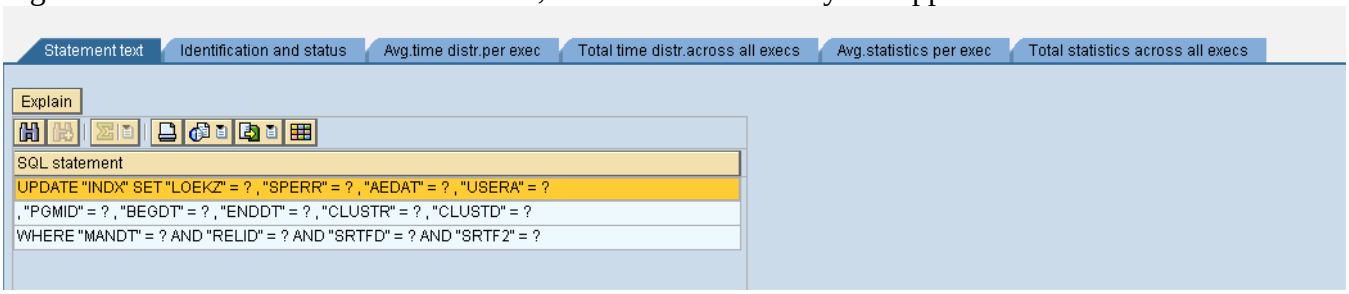


Figure 137: ST04 statement

The ABAP and the run processes for the application needs to be examined to determine why there are so many concurrent processes updating the same row in the database, or whether they hold the locks for too long.

8.5. Examples of ST04 statement cache analysis

8.5.1. Predicates do not match available indexes

SAP ECC, as a transaction processing system, generally uses simple SQL that can be executed using index access on a single table, or nested loop join on multiple tables.

The most common problem with inefficient SQL is when the local predicates (selection criteria in the SQL) do not match the available indexes well. In this case, DB2 will choose the best access path it can find for the predicates in the SQL. This access path may still be rather inefficient.

An SQL statement scanning a small in-memory table will often be indicated as a CPU usage problem, rather than having long total response time. Here, the statement cache has been sorted by CPU, and we look for statements with high 'Avg. CPU time'.

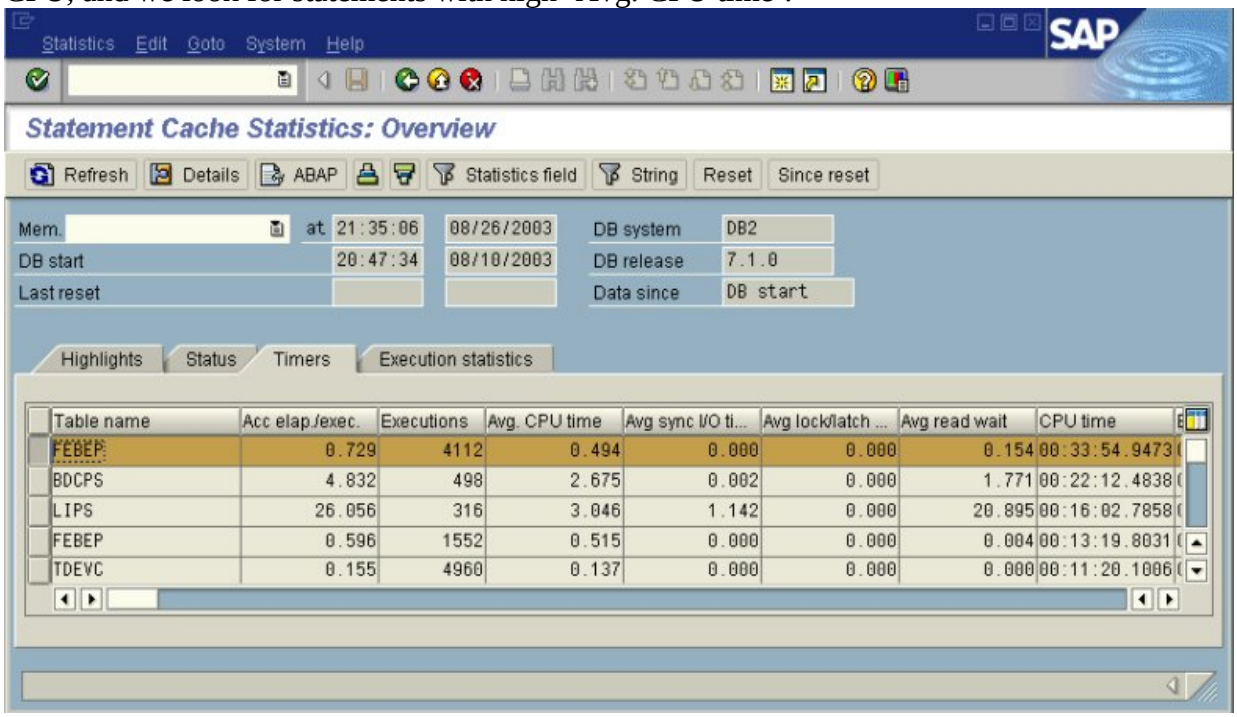


Figure 138: FEBEP - ST04 sorted by CPU

After sorting the ST04 statement cache by CPU, the statement accessing FEBEP shown in Figure 139 was near the top of the list. We want to determine whether it is efficiently coded, or if it can be improved.

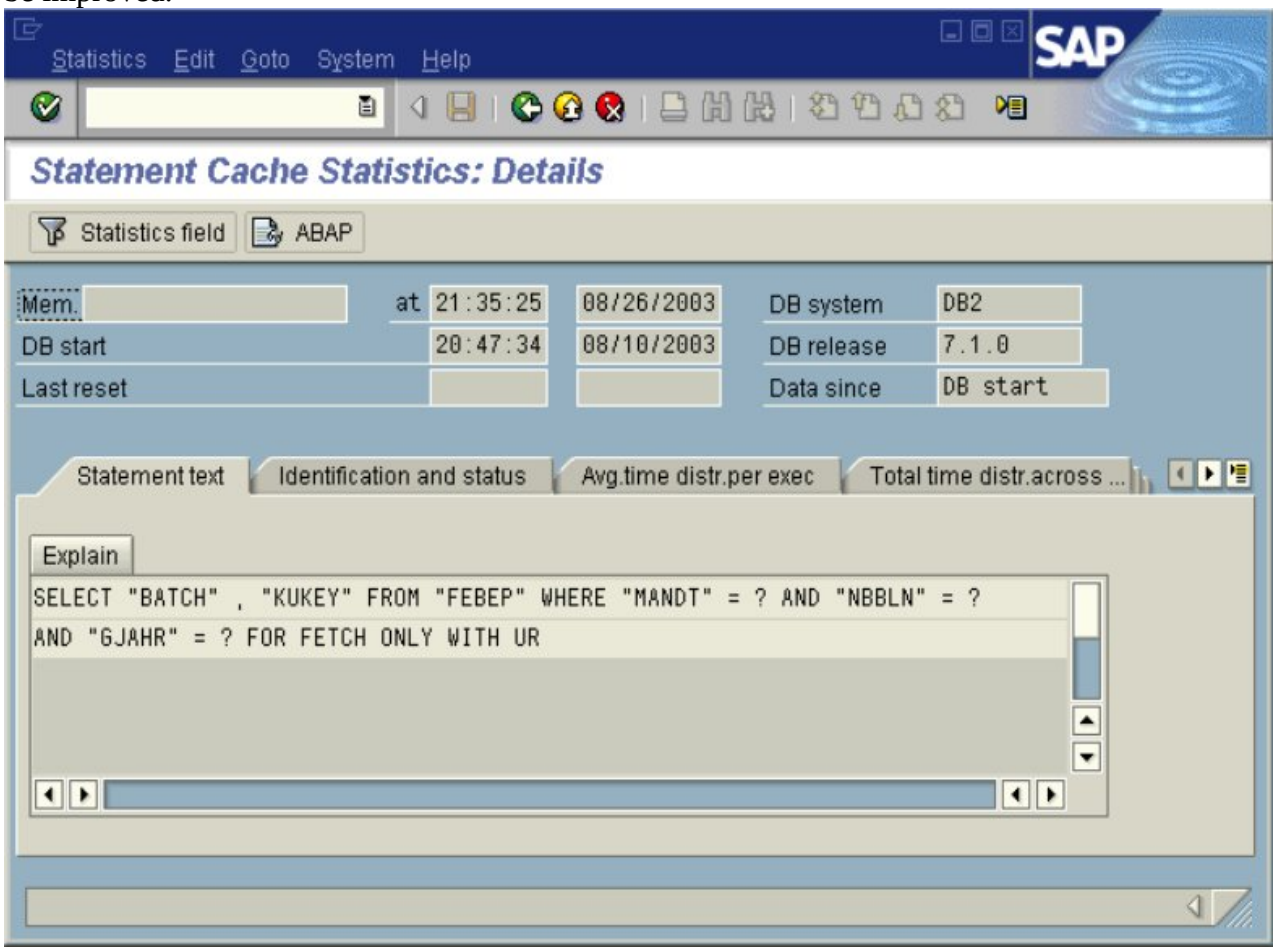


Figure 139: "details" display of FEBEP statement from ST04 cached statement

Make note of the local predicates (MANDT=, NBBLN=, and GJAHR=) for later reference.

In Figure 138, the statement accessing FEBEP uses almost 500 ms CPU per execution, which is very high. It could be normal for a statement that retrieves many rows. Select the statement, and press Details to see more information about the statement.

In the "details" display, select the "Avg. statistics per exec" tab to check the per-execution statistics for the statement, and see that it is performing 17,886 getpages per execution, and it processes (returns) less than one row (0.62) per execution. An efficiently indexed R/3 statement usually needs at most a few getpages per row processed. "Rows exam/exec" is very high, so we know local predicates are being applied on table rows.

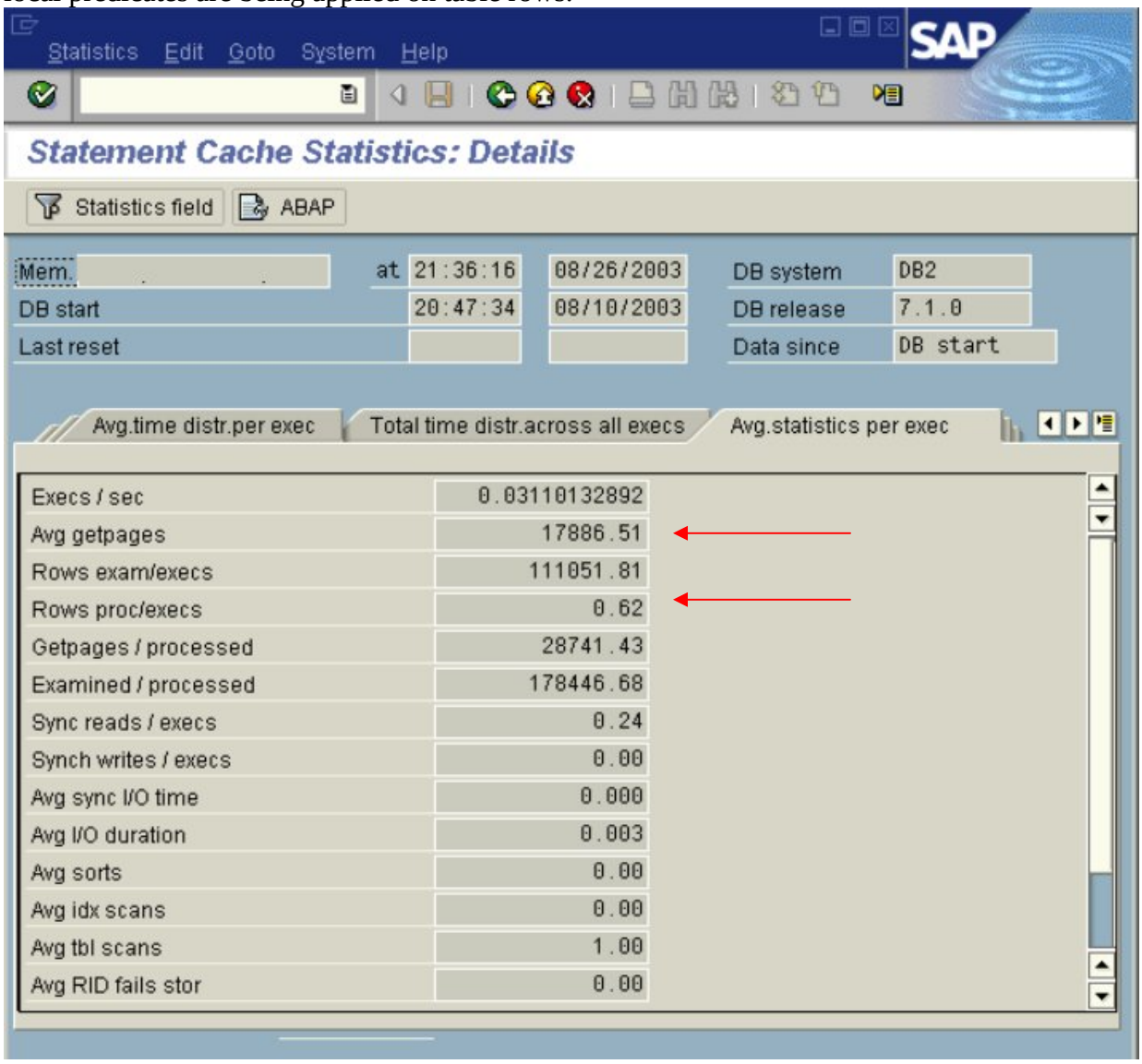


Figure 140: ST04 cache "details" display of execution statistics

Next, from the “statement text” tab in “details”, explain the statement to check the access path being used. The explain shows that tablespace scan is used. All rows were read from the table, and all local predicates applied on table rows.

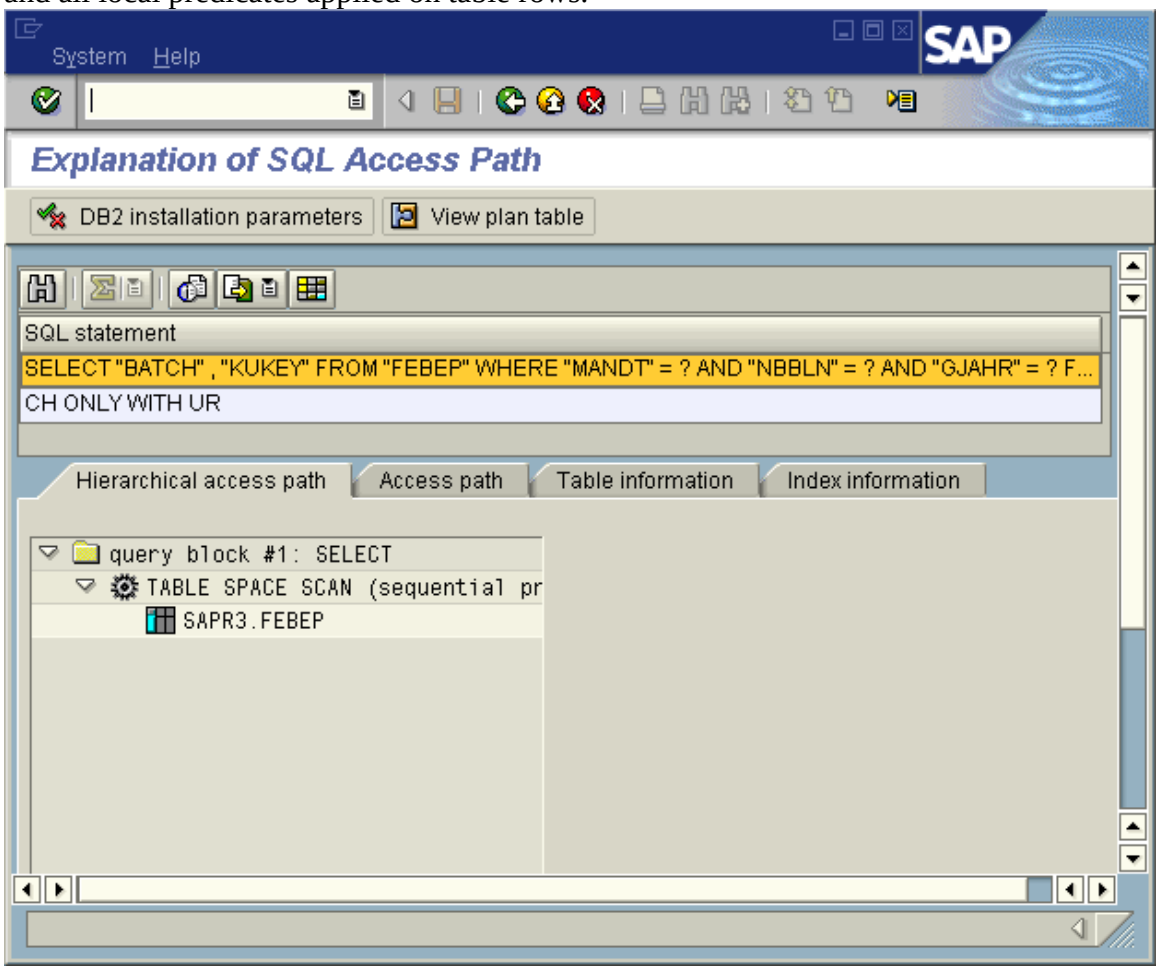


Figure 141: Explained statement from ST04 cached statement details

Use the explain 'Index information' tab to check to see if the local predicates (MANDT=, NBBLN= and GJAHR=) are in an index on FEBEP.

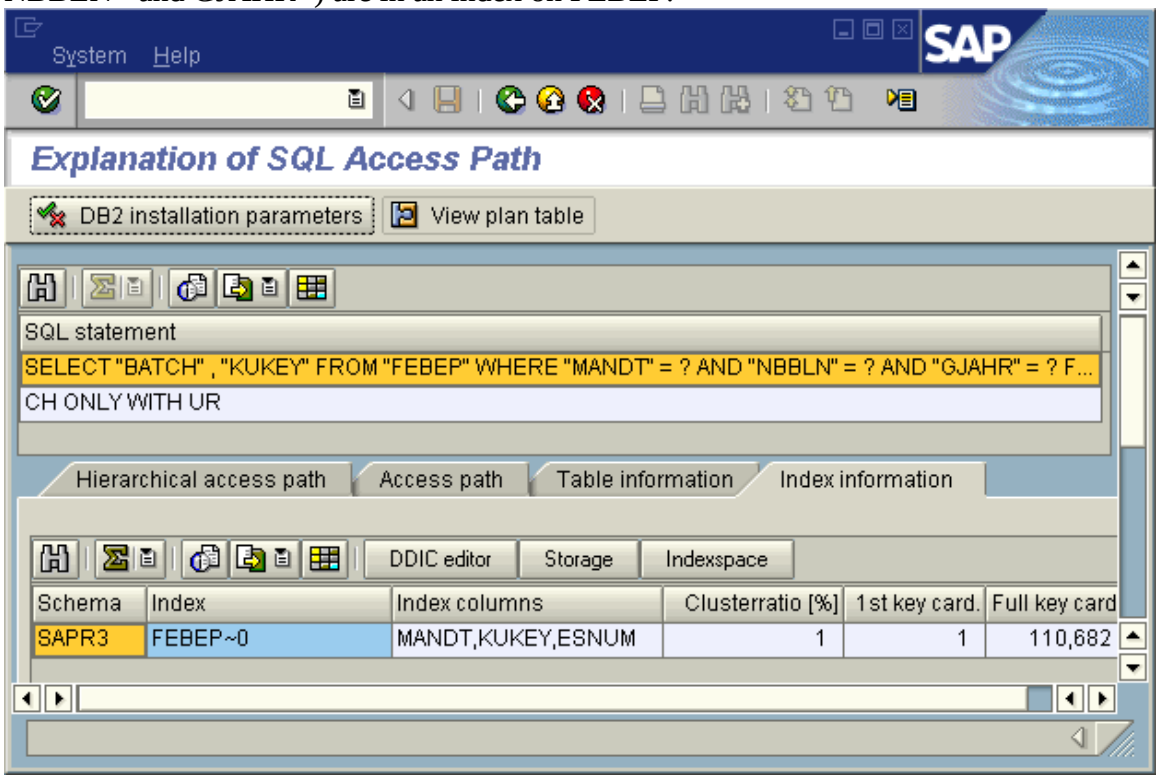


Figure 142: FEBEP - ST04 Index information

In Figure 142, note that neither NBBLN nor GJAHR is in the one index, so the index will not be used to access the table. MANDT is in the index, but has cardinality 1, so it probably does not filter.

Now check if either predicate column has high cardinality. Cardinality is the number of unique values. If either column has high cardinality, it would make a more efficient way to access the table with an index. (The exception being when the data is unevenly distributed, as shown in Section 8.3.)

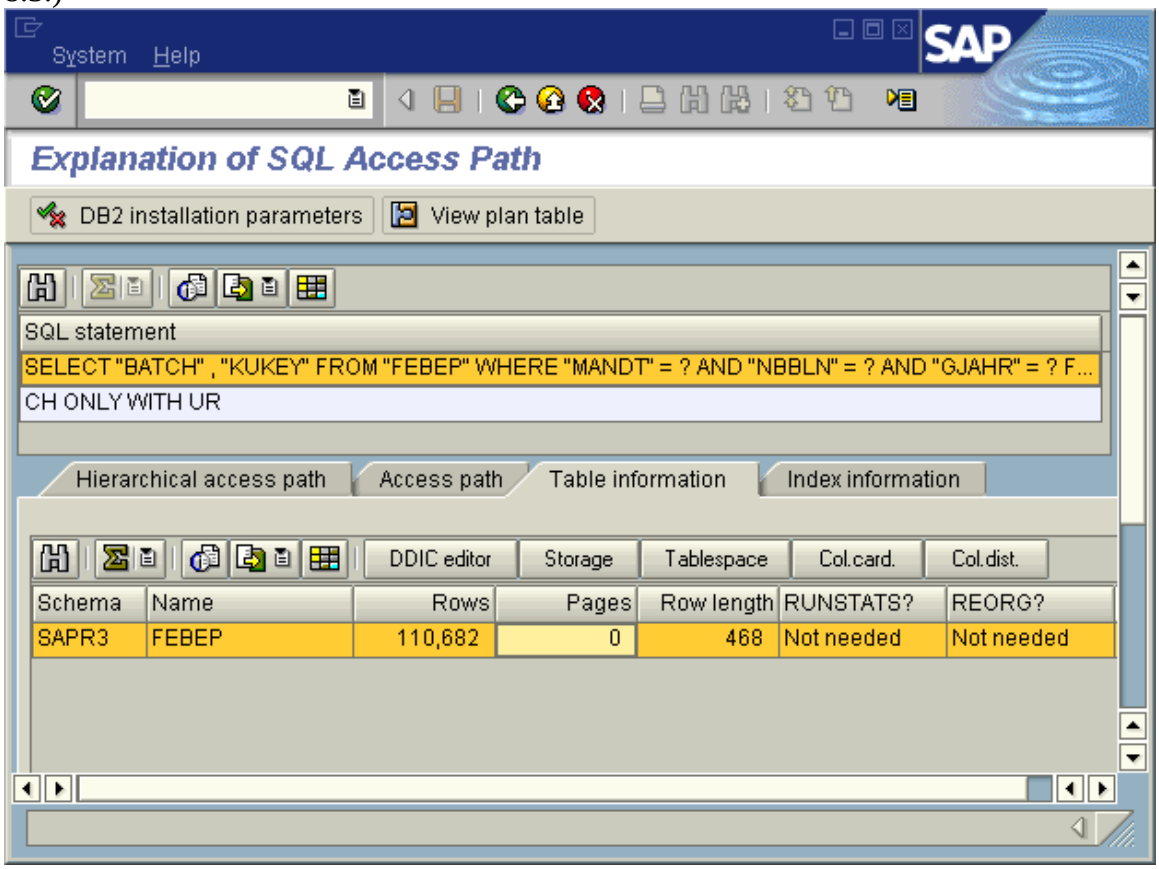


Figure 143: FEBEP - Table information

Choose the table, and press the 'Col card.' button, to see column cardinality statistics.

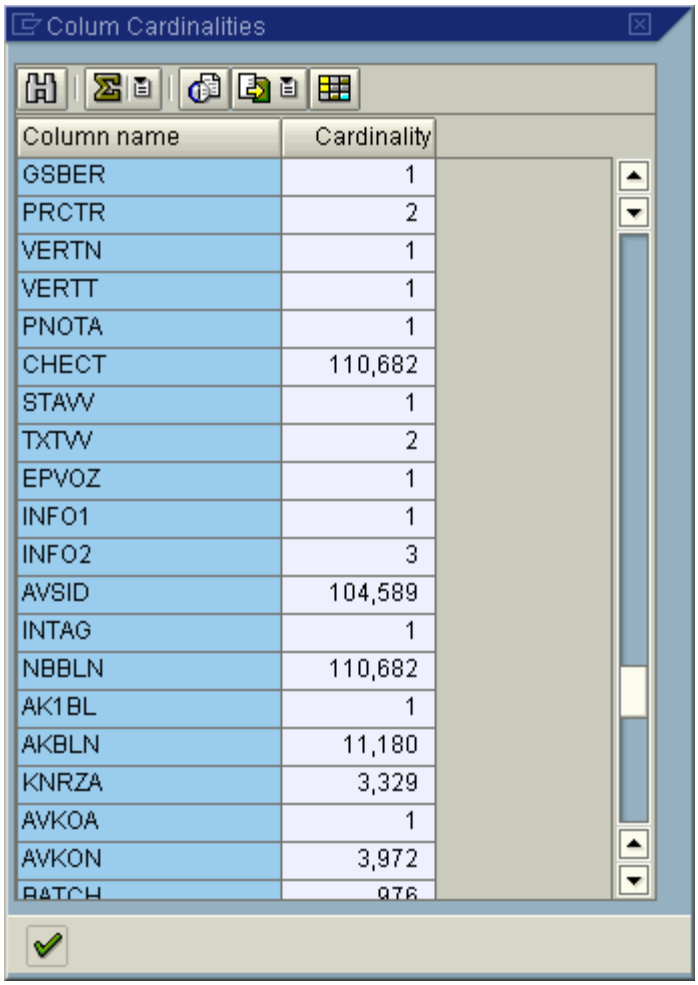
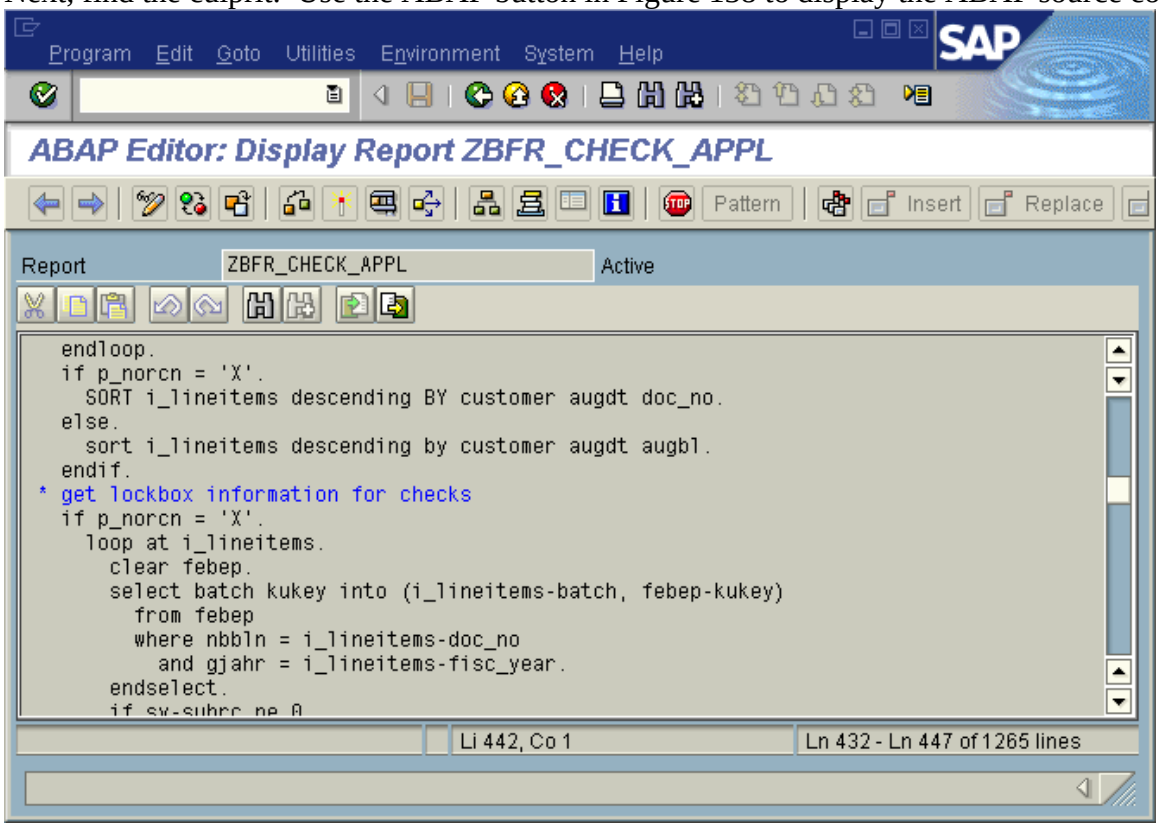


Figure 144: FEBEP column cardinality statistics

In Figure 144 we see that NBBLN has 110,682 unique values, so it will filter well, as Figure 143 showed 110,682 rows in the table. Each row has a different NBBLN value.

Next, find the culprit. Use the ABAP button in Figure 138 to display the ABAP source code.



Please see Section 7.5.2 regarding the process for evaluating when to add new indexes. Since the program is a custom program (Z*), the first action is always to investigate whether the program could be modified to match the available index. The next action would be to review if this is a problem of misuse of the data model. The last action would be to add a new index that matches the statement.

8.5.2. Impact of REOPT(ONCE)

As described in Section 8.3, the default method of preparing statements with parameter markers can cause access path problems in some situations, such as when there is data skew, and filtering depends on the values used in the SQL.

To address this issue, the REOPT(ONCE) option was created, which allows a statement to be prepared with values, but still shared by all dialog steps that execute the SQL, regardless of the runtime parameters. However, if the parameters used with first execution of the statement are not representative of the normal execution, this can cause DB2 to choose the wrong access path.

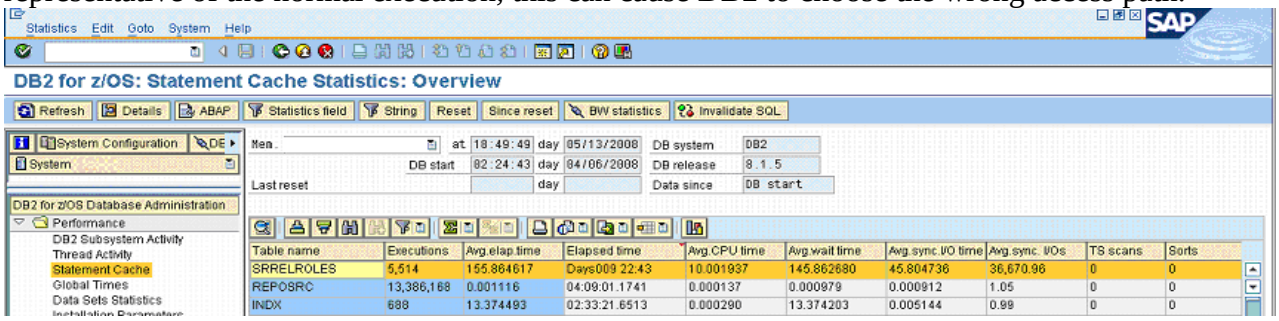


Figure 145: SRRELROLES

In Figure 145, we have sorted the cache by elapsed time, to find the statements with the most impact on response time. Each select on SRRELROLES takes 155 ms, of which 145 ms is wait. We need to look at the statement details to see why there is so much delay.

Select the statement, and press the “Details” button.

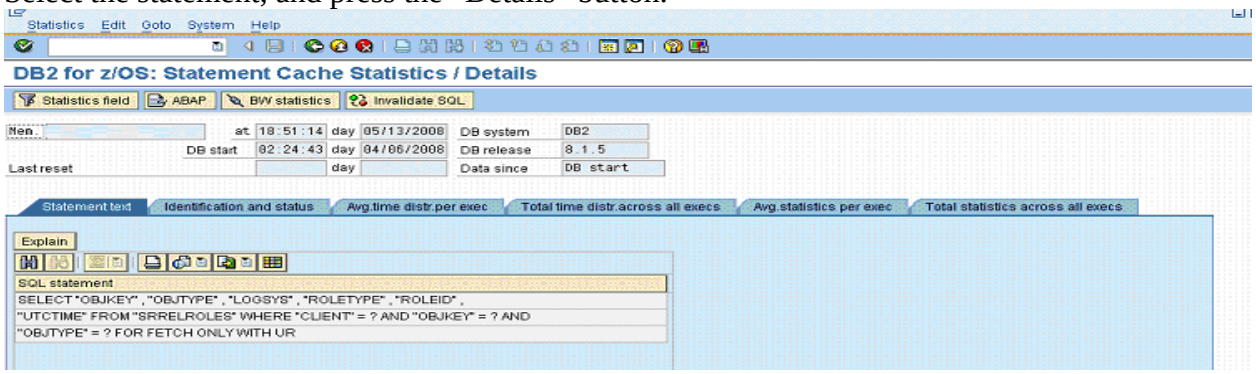


Figure 146: SRRELROLES statement text

Local predicates are CLIENT=, OBJKEY=, OBJTYPE=.

Press the “Avg statistics per exec” tab.

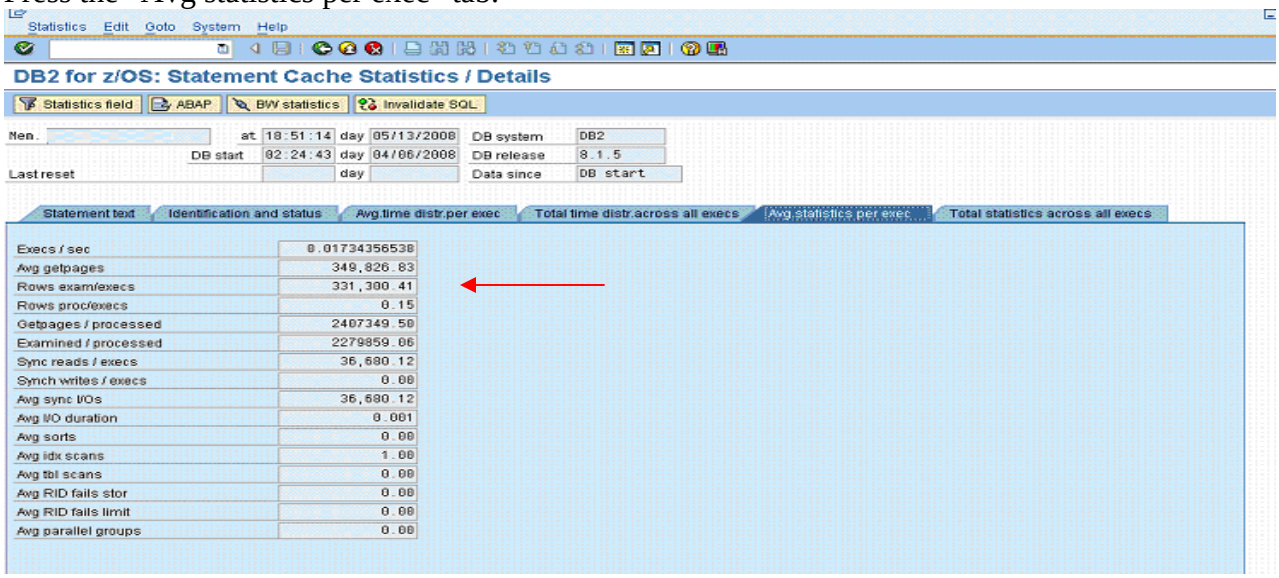


Figure 147: SRRELROLES statistics

Figure 147 shows each execution processes over 331K rows, to return less than one row. Local predicates are not being evaluated in the index, but on the table (high Examined/processed).

Go back to the “statement text” tab, and press “Explain”, then press “Hierarchical Access Path” tab.

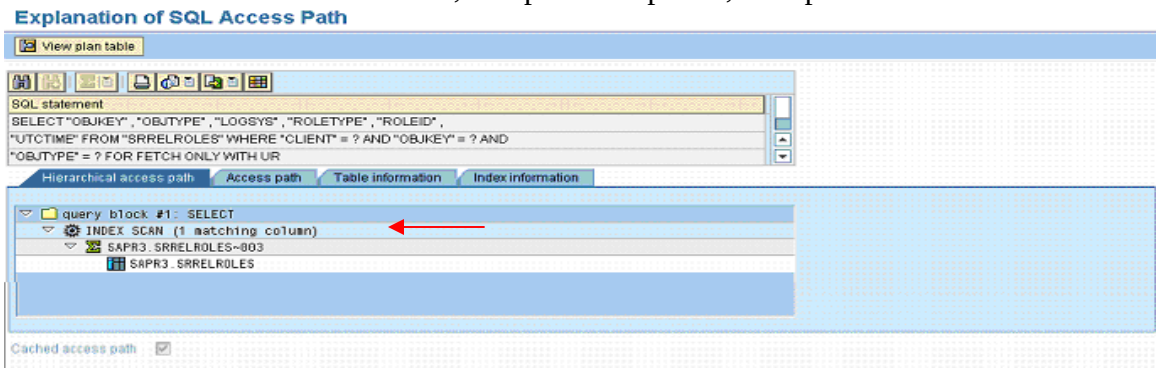


Figure 148: SRRELROLES Hierarchical Access Path

Access is one matching column in index SRRELROLES~003. There could be index screening on other columns, we need to check the indexes.

Press the “Table information” tab, then select the table and press “Col card”, to display the column cardinalities, which we will need for index evaluation.

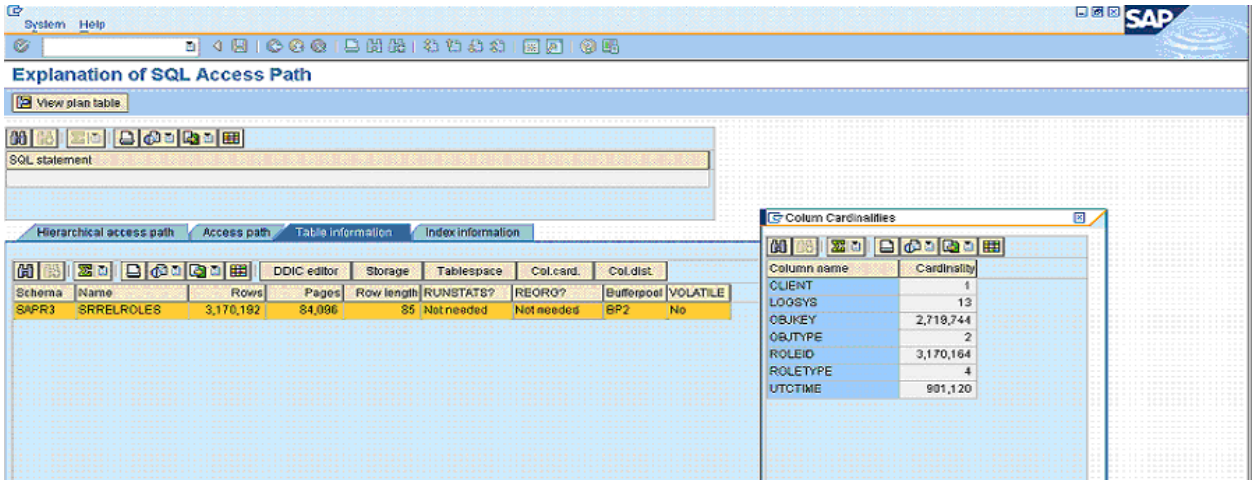


Figure 149: SRRELROLES table column cardinality

One can also display the column distribution information (i.e. DB2. FREQVAL) by selecting the table, and pressing “Col dist.”.

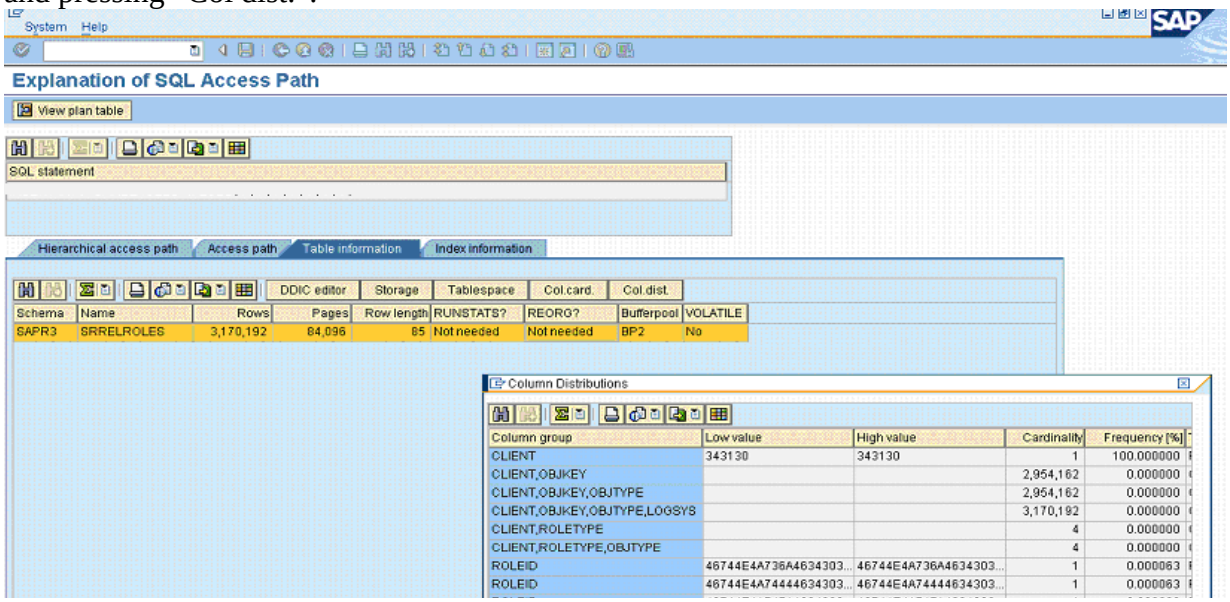


Figure 150: SRRELROLES column distribution

Press the “Index information” tab to see the indexes.

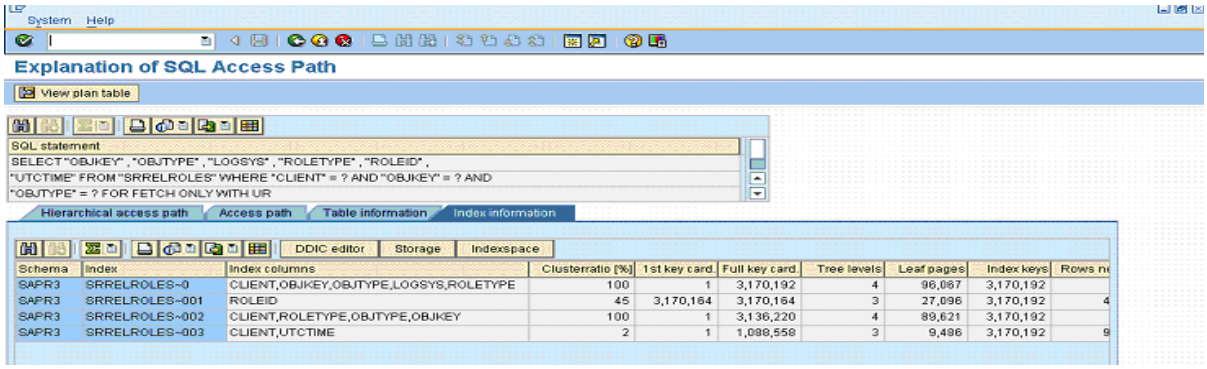


Figure 151: SRRELROLES indexes

And now we see something surprising. The index that is being used (SRRELROLES~003) matches only CLIENT, while there is an index (SRRELROLES~000) that matches CLIENT, OBJKEY, and OBJTYPE. And, from Figure 149, we know that OBJKEY cardinality is over 2 million, so the OBJKEY= local predicate is strongly filtering.

Why did DB2 choose the ~003 index? The clue is in the cardinality of client, which is 1, and the only value is “410”. If this statement is not run from the productive client, and if DB2 is configured with REOPT(ONCE), then the values passed to the DB2 will include CLIENT=xxx, where xxx is not the productive client (410). Using this, DB2 will check the CLIENT value passed from SAP (xxx) against the catalog statistics (410), and determine that matching the column CLIENT alone provides all the filtering needed. And, whenever the statement is run from client xxx, this is correct. But if the statement is later run from client 410, then the CLIENT column provides no filtering, and all the table rows must be read.

There are two clients (000 and 001) delivered with an SAP system, and other clients are created as part of the customization process. SAP programs that can run in 000 or 001 or in the productive client can encounter this problem.

SAP note 1008334 describes the situations where REOPT(ONCE) can cause access path problems, and how to fix them.

As a short term fix, one can use the ST04 statement cache “invalidate SQL” button (see Figure 145) to invalidate the statement so that it is re-optimized when next run, or one can execute ‘RUNSTATS REPORT NO UPDATE NONE’ on the table if using a version of SAP that does not have ‘invalidate SQL’ in ST04.

8.5.3. Incorrect use of SAP data model

As mentioned in Section 7.4.2, data is often de-normalized and redundantly stored in SAP tables, in order to optimize performance. When we have a custom APAB program using SAP tables in a way that is not supported by indexes, our first approach in analyzing the problem is to determine whether the program is looking in the wrong place for the data. At first glance, a situation where the data model is

not being used correctly may appear to require a new index to optimize the program, but there is a different and better way to solve the performance problem which we will see below.

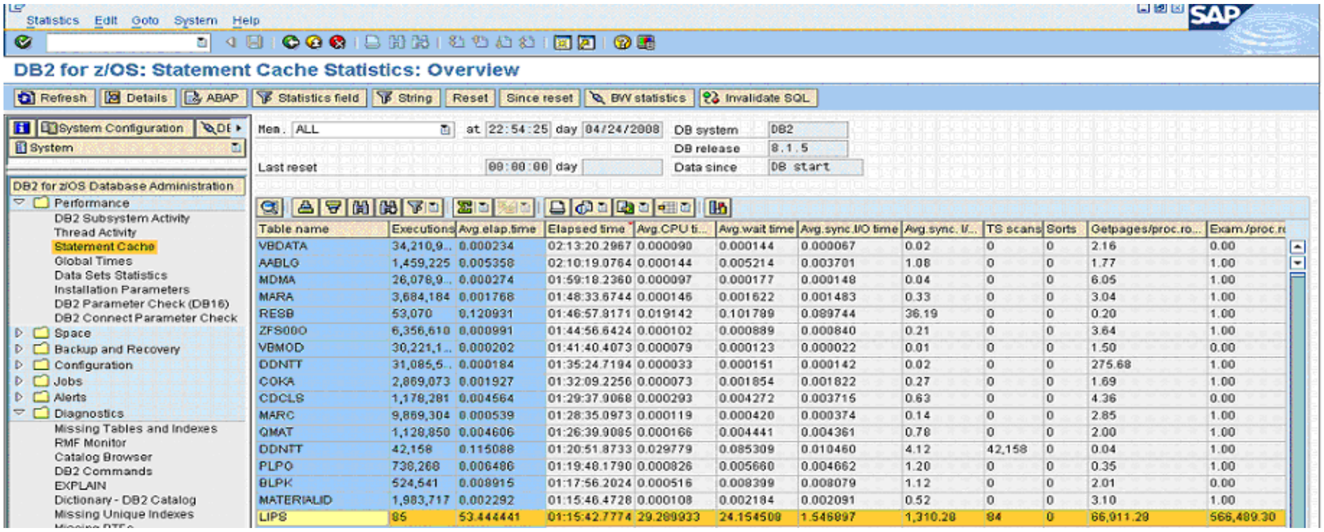


Figure 152: LIPS Statement Cache

Here, we have ordered the statement cache by elapsed time, to find high impact statement. The statement on LIPS stands out – it has high "getpages/proc row" and high "examined/proc row". This means that DB2 must search many pages for the result (getpages/row) and that DB2 must apply local predicates on the table (rows examined/row processed).

Select the line and press "Details" to see the statement.

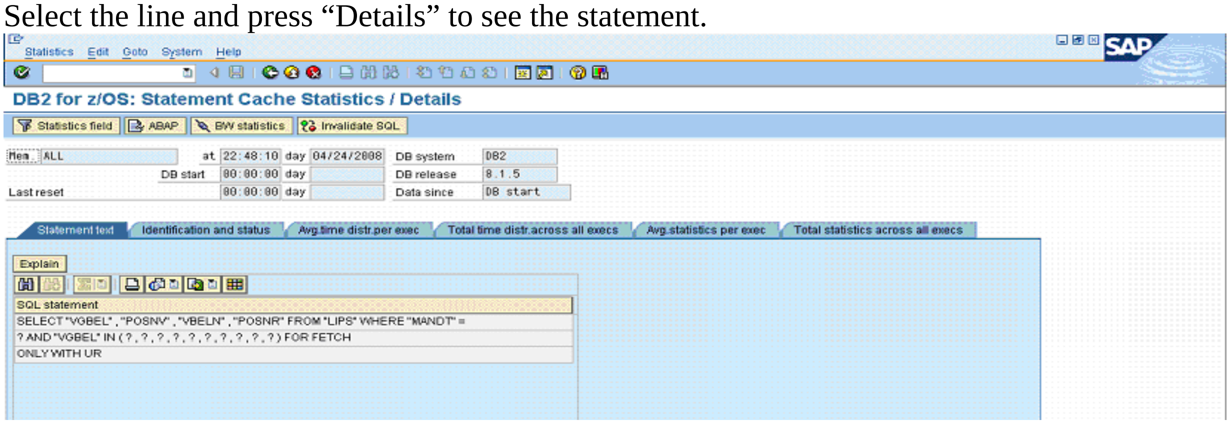


Figure 153: LIPS statement

It looks normal so far. Statement has local predicates MANDT= and VGBEL IN.

Press the "ABAP" button tab to see the program. We need to check if it is SAP or custom.

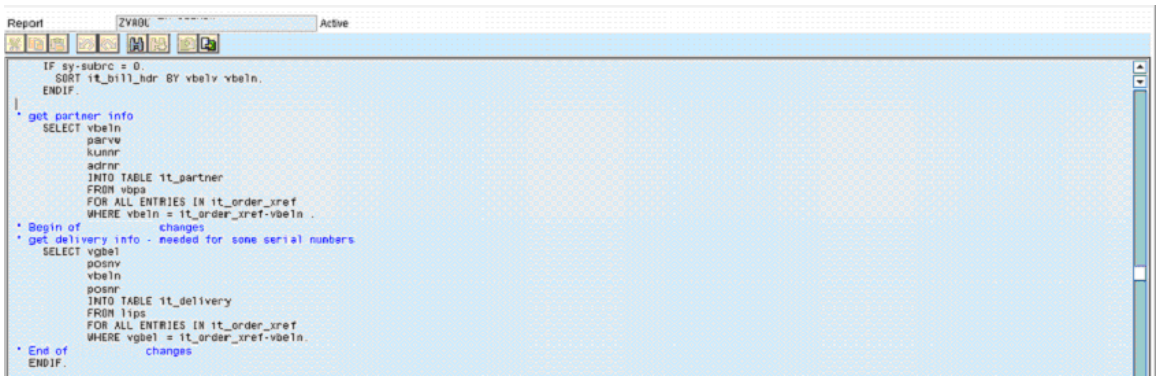


Figure 154: LIPS program

Program starts with a Z, so it is custom. One can also display the program attributes to see whether SAP or some other user created it.

We need to explain it to see what DB2 is doing. Press the "Explain" button in Figure 153.

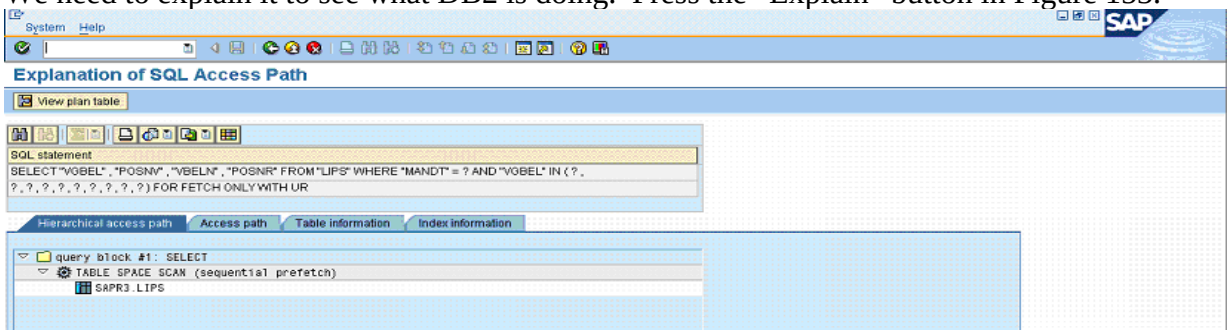


Figure 155: LIPS explain

Figure 155 shows DB2 is scanning the entire table. We press "Index Information" to check indexes and compare them to local predicates.

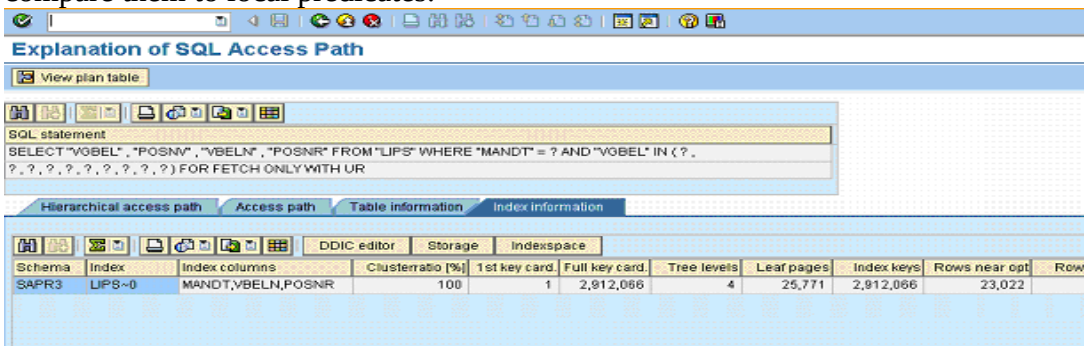


Figure 156: LIPS Index

VGBEL does not appear in any index. The only other local predicate is MANDT=, and we see in Figure 156 that MANDT has cardinality of one (1st key card), so MANDT= probably will not filter. DB2 reads the entire table.

Check the chart in Section 7.5.2. Note that when we have custom code and SAP tables, we need to check the use of the data model.

SAP note 185530 describes this problem, and proposes the change to ABAP to use the SAP tables and indexes correctly.

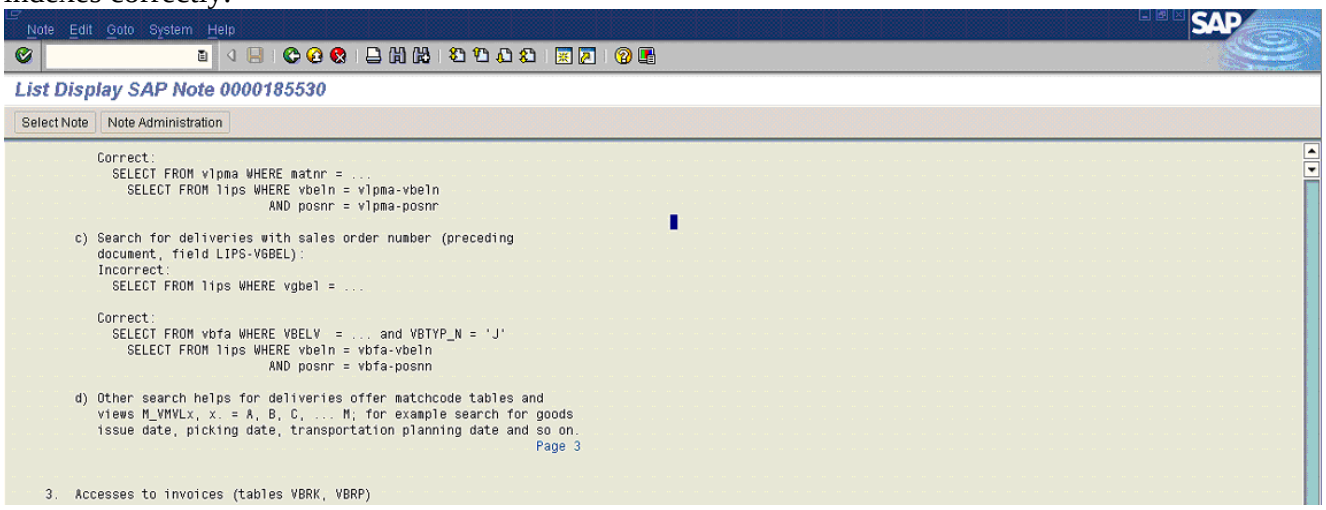


Figure 157: NOTE 185530

At this point, we send the program back to development, so they can change the program as suggested in the SAP note. The change will allow the program to read LIPS using an index, rather than scanning the entire table.

8.5.4. Index-only access

As of DB2 V8, it is possible to have index-only access with SAP on DB2 for z/OS. The statement statistics in this case can be a bit misleading.

Since there are no rows examined (DB2 never had to read the table), the the ratio “Examined/rows proc” is 0. With index-only access, one has to check the “Getpages/proc row” ratio to see whether the statement is efficient or not, as shown in Figure 158.

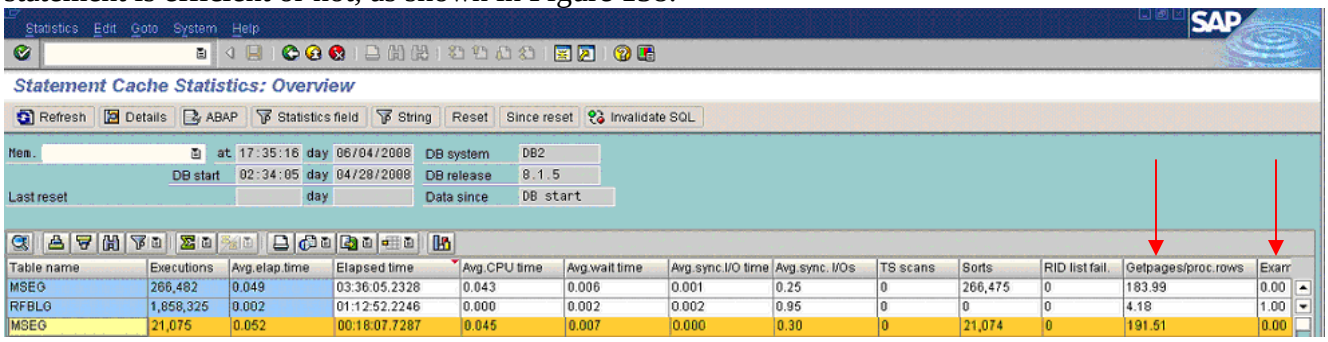


Figure 158: ST04 index-only access

But, since “Examined/proc rows” can also be zero if there are no rows processed, we also need to check the number of rows processed per execution, to confirm whether this is a statement that returns no rows, or is index-only access. This is show in Figure 159.

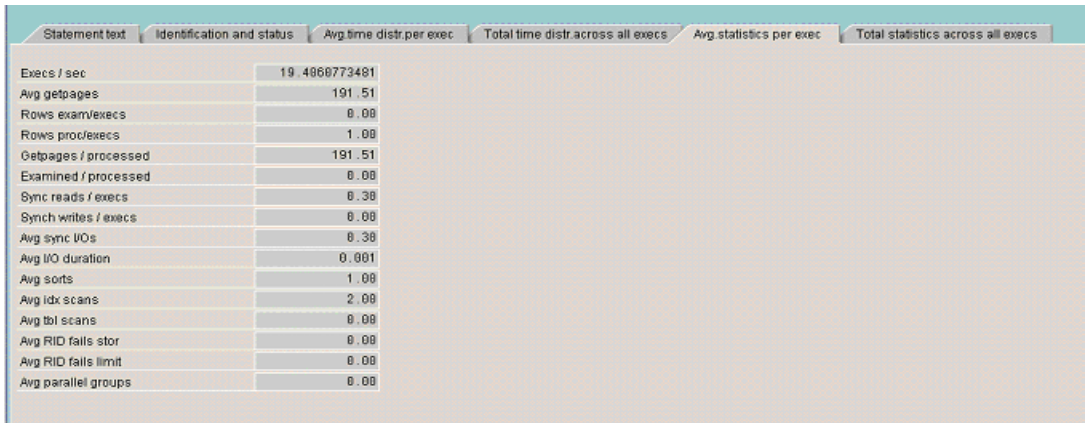


Figure 159: ST04 index-only statistics per exec

"Rows exam/execs" is 0, so this is index only access. "Rows proc/execs" is 1, so one row is returned on each execution.

But note that index-only is not enough to be efficient access. This statement does almost 200 getpages per row, which is high. With all predicates evaluated in an index, a single row fetch might be 4-5 getpages/exec, and a two table join might be 8-10 getpages per exec.

Even though this is index-only, we would explain the statement, and evaluate the access path and indexes, as shown in Sections 8.5.1 and 8.5.3.

8.5.5. Column order with custom indexes

When creating a custom index to support a business process, the order of the columns in the index can be important. There are two key issues:

- Range predicates (GT, GE, LT, LE, BETWEEN, LIKE) on some of the index columns, and
- Non-uniform distribution of data (skew) on one or more of the columns.

Here is an example of a statement that is frequently run, and one of the top statements in elapsed time. Each execution takes 177 ms. (This is from a 6.40 system where the time units in ST04 are seconds).

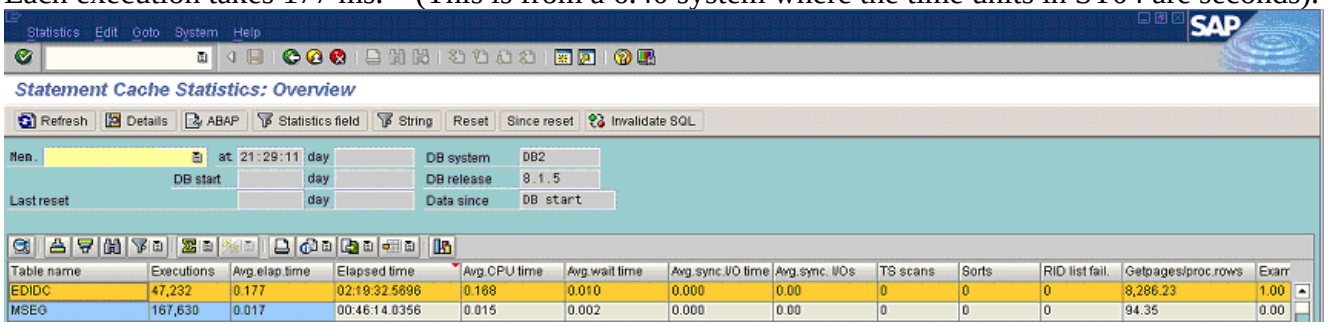


Figure 160: EDIDC many getpages per row

Select the statement and press details (Figure 161), and we see that getpages per row is high (over 26K), because there is seldom a row returned (0.03 per exec), but there are 797 getpages per execution, which is high.

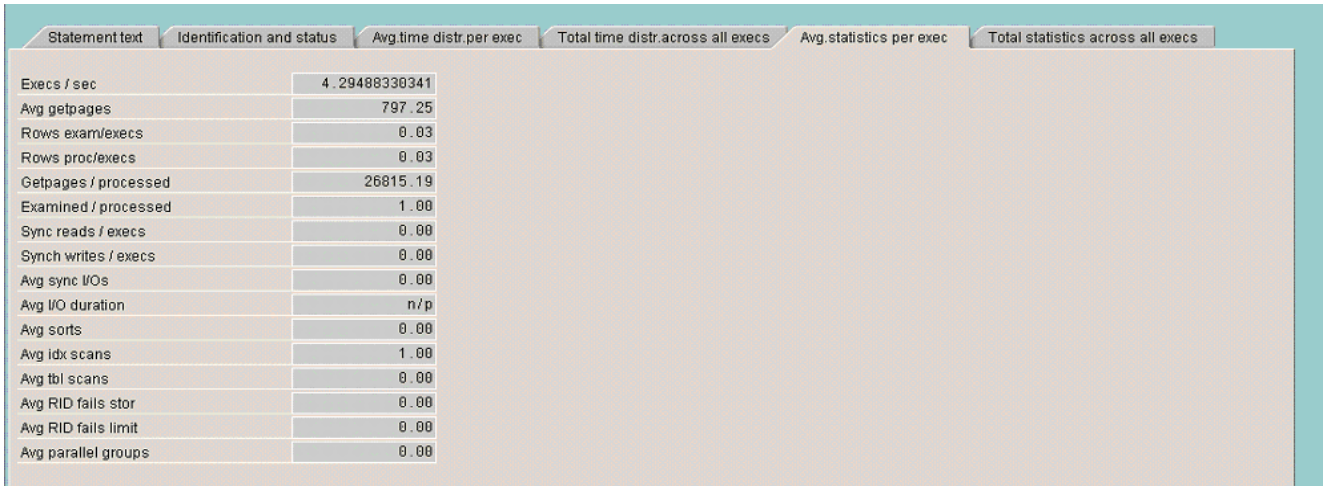


Figure 161: EDIDC statistics per execution

We explain the statement and check the access path and statement text. There are local predicates on MANDT, STATUS, MESTYP, and CRETIME.

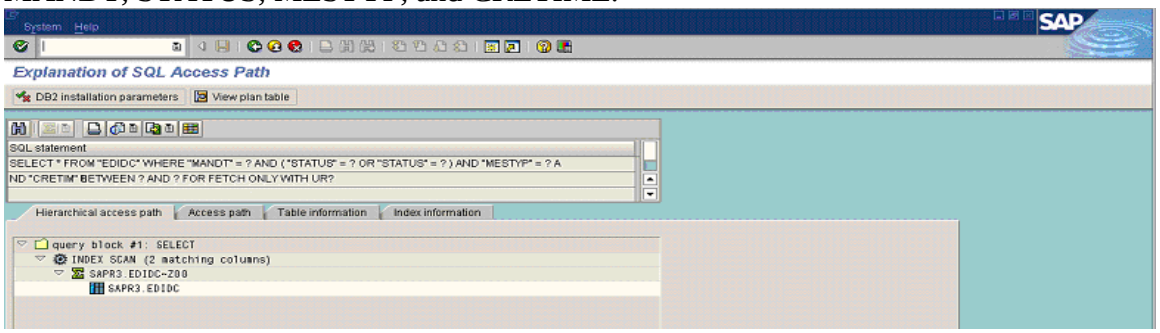


Figure 162: EDIDC explain

Press the “Index information” tab, to check the indexes in Figure 163, we see the problem. The column with the range predicate (CRETIM BETWEEN) is the second column in the index (MESTYP, CRETIM, STATUS). DB2 cannot do index matching access for STATUS, or for any index column on the right of a column with a range predicate. DB2 is doing index screening, which is more efficient than evaluating the local predicates in the table, but less efficient than matching index access. (We can tell it is index screening, since rows examined per exec is the same as rows processed per exec – no local predicates were evaluated on the table).

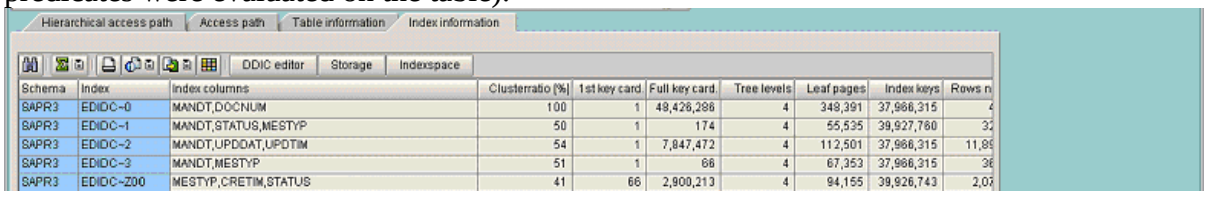


Figure 163: EDIDC indexes

If the order of the index columns were (STATUS, MESTYP, CRETIM) or (MESTYP, STATUS, CRETIM) then DB2 would be able to do three column matching access.

With the current index column order, DB2 does index screening on STATUS, which is more efficient than examining the table rows, but less efficient than index matching access.

Now we need to make the choice of whether to make the index (STATUS, MESTYP, CRETIME) or (MESTYP, STATUS, CRETIME). By default, DB2 collects column distribution (FREQVAL) data only on the first column of an index, so if one puts a column with skewed data as the first column, then there is not a need for custom RUNSTATS. The default RUNSTATS, together with ABAP HINTs or REOPT(ONCE) BIND option will enable DB2 to determine that there is skew, and choose an appropriate access path.

If there is a column with data skew that is not the first column in the index, then COLGROUP FREQVAL statistics must be gathered.

In order to evaluate if there is skew, we use SPUFI with a query like:

```
select STATUS, count(*) from EDIDC group by STATUS order by 2 desc;
```

Figure 164: query to check for skew

And we will get an answer like:

```
SELECT STATUS, COUNT(*) FROM EDIDC
GROUP BY STATUS
ORDER BY 2 DESC ;
     +---------------------------------+
     |    STATUS     |                 |
     +---------------------------------+
  1_|             X |         1006307 |
  2_|               |           63146 |
     +---------------------------------+
```

Figure 165: STATUS counts

There are only two values of STATUS, and one occurs much more frequently than the other. The data is skewed. When a statement is executed with WHERE STATUS = ‘ ‘, then the STATUS column will filter well. But if a statement is executed with WHERE STATUS = ‘X’, then the STATUS column is not useful for filtering. By having this information, DB2 can make a better choice which index to use.

So, we will create the index as (STATUS, MESTYP, CRETIM). Putting STATUS first in the index does not make the SQL execution more efficient, but it makes it easier to manage the table, since custom RUNSTATS are not needed to gather the FREQVAL data, DB2 will gather FREQVAL by default.

8.5.5.1. Considerations for column order of custom indexes

Since index processing is affected by the type of local predicates, if you need to add a new index, please use the following general order:

- Filtering columns with equals predicates
- Filtering IN predicate or a filtering OR predicate
- Filtering range predicates (GT, LT, BETWEEN, etc)
- Other predicates (second IN, or range, or NE)

Here we are having an exception to these suggestions (placing column with IN before =), in order to gather the statistics on the skewed STATUS column without custom runstats.

After we change the column order to (STATUS, MESTYP, CRETIM) then the statement runs much more quickly – 6 ms per execution in Figure 166, compared with 177 ms per execution in Figure 160.

Table name	Executions	Avg.elap.time	Elapsed time	Avg.CPU time	Avg.wait time	Avg.sync.I/O time	Avg.sync. I/Os	TS scans	Sorts	RID list fail.	Getpages/proc.rows	Exam
EDIDC	34,376	0.007	00:03:51.1240	0.006	0.001	0.000	0.00	0	0	0	1,344.82	1.00

Figure 166: EDIDC wth new index

Keep in mind that even a statement with an elapsed time of 100 or 200 ms can have a significant impact on response time, if it is executed frequently enough. This is a summarized trace of the IDOC processing with the original index design, and you can see that the selects on EDIDC were about 60% of the total DB time, so fixing the problem would reduce the DB request time by over 50%.

Table Overview Edit Goto System Help

Summarized Table Overview

Transaction	Table Name	Statement	Accesses	Records	Time	Percent
	ADRC	SELECT	1	1	4,292	0.0
	AKB_PRODCOMP	SELECT	2	2	7,892	0.0
	ATAB	SELECT	70	140	118,154	1.0
	AUSP	SELECT	88	88	137,089	1.0
	CABN	SELECT	1	1	1,303	0.0
	CEPC	SELECT	1	1	1,378	0.0
	D020S	SELECT	1	0	1,234	0.0
	EDI40	SELECT	1	1	2,046	0.0
	EDIDC	SELECT	51	2	8,222,701	68.0
	INDX	SELECT	3	0	3,997	0.0
	INDX	UPDATE	1	0	1,091	0.0
	INDX	INSERT	1	1	1,959	0.0
	MAKT	SELECT	10	93	17,593	0.0
	MARA	SELECT	216	304	396,324	3.0
	MARC	SELECT	80	163	255,348	2.0
	MARD	SELECT	243	1,902	486,321	4.0
	MARV	SELECT	1	1	1,034	0.0
	MBEW	SELECT	79	159	248,610	2.0
	MCH1	SELECT	138	138	253,335	2.0
	MCHA	SELECT	139	138	364,284	3.0
	MCHB	SELECT	138	138	281,267	2.0
	NRIV	SELECT	1	1	9,415	0.0
	RESB	SELECT	5	466	68,801	1.0
	RESB	UPDATE	92	92	155,908	1.0
	RESB	INSERT	3	3	19,039	0.0
	RKPF	SELECT	2	2	5,377	0.0
	RKPF	UPDATE	1	1	1,359	0.0
	TRDIR	SELECT	100	100	241,360	2.0
	VARI	SELECT	50	0	106,225	1.0
Total			1,774	5,276	12,089,512	100.0

Figure 167: EDIDC in summarized trace

8.5.6. Logical row lock contention

Row lock contention (lock/latch wait in ST04) is almost invariably an application issue, which usually cannot be fixed at the DB level. When you encounter timeouts or deadlocks, one should open a message to SAP so they can investigate the cause.

This is common on statistics tables and ledger (e.g. general, special) tables. These are tables containing rows where information is being consolidated or aggregated. Lock contention is caused by the design of

the application and the business data. For instance, if all sales were posted to a single “cost of goods sold” account, then that row would become a constraint in the database when the transaction volume reaches a certain level. Below the critical level, it is not a problem.

Depending on the root cause, the actions required to fix row lock contention may be

- changes to SAP settings (“late exclusive material block”, “posting block”, etc) ,
- ABAP coding (perform change SQL just before commit work),
- business structure in SAP tables (e.g. more granularity in chart of accounts),
- sorting or grouping input rows to avoid contention during processing.

If it cannot be fixed in one of these ways, then test to determine the level of parallelism that gives the maximum throughput (batch, updates, etc) and configure the system to keep parallelism under this level, to manage locking contention.

The two system indicators for lock contention are ST04 “lock/latch” time being high, and change SQL statements in the ST04 statement cache with long elapsed time. In order to prove that it is a lock problem, one would need further information.

```
Instance      : d660_GST_00
Statistic file: /usr/sap/GST/DVEBMGS00/data/stat
Analyzed time : 12/19/2001/22:30:00 - 12/19/2001/22:40:29   (with further selection criteria)

                                                  Response Memory   Wait     CPU      DB req.  Load/Gen kBytes   Phys. db
 End time Tcod   Program  T Scr. Wp User           time(ms) used(kB) time(ms) time(ms) time(ms) time(ms) transfer changes
-------------------------------------------------------------------------------------------------------------------------
|22:35:41 VL02   RSM13000 U 3000 45 0656         | 13,173 |  4,510 |  5,728 |    970 |  6,561 |     58 |1,242.6 |    119 |
|22:35:41 VL01   RSM13000 U 3000 48 0983         | 13,376 |  4,648 |  5,324 |  1,470 |  6,606 |     47 |1,284.3 |    210 |
|22:35:42 VL02   RSM13000 U 3000 57 0609         | 13,540 |  4,510 |  5,643 |    970 |  6,913 |     44 |1,242.6 |    119 |
-------------------------------------------------------------------------------------------------------------------------

  Table accesses sorted by time             (list might be incomplete)
-------------------------------------------------------------------------------
|                      -------- Number of rows accessed ----------             |
| Table name                Total Dir. reads Seq. reads    Changes  Time (ms)   |
| -------------------- ---------- ---------- ---------- ---------- ----------  |
| TOTAL                       162          1        136         25      6,470   |
|                                                                              |
| GLT0                          2          0          0          2      6,130   |
| TRFCQOUT                     11          1          1          9        283   |
| CIF_IMOD                    135          0        135          0         32   |
| VBBE                          7          0          0          7         15   |
| MSEG                          7          0          0          7         10   |
-------------------------------------------------------------------------------
```

Figure 168: STAT record with long GLT0 change times

While the job is running, one can use ST04 Subsystem activity to display currently held locks. Section 8.4.3 showed an example of using ST04 statement suspension times in diagnosing locking problems.

For systems which do not have RFCOSCOL ST04 with its suspension time breakdown, here is a DB2PM “LOCKING REPORT LEVEL(SUSPENSION LOCKOUT) ORDER(DATABASE-PAGESET)” report processing an IFCID 44,45 trace and showing row lock contention on rows in GLT0. Lock contention is counted in the “LOCAL” counter for an object.

```
     MEMBER: N/P                                                                    TO: NOT SPECIFIED
   SUBSYSTEM: DBW0                          ORDER: DATABASE-PAGESET          INTERVAL FROM: 09/16/99 10:48:28.35
 DB2 VERSION: V5                                 SCOPE: MEMBER                        TO: 09/16/99 10:55:22.29
)
                                                                --SUSPEND REASONS-- ---------- R E S U M E   R E A S O N S ----------
 DATABASE            --- L O C K   R E S O U R C E --- TOTAL    LOCAL  GLOB.  S.NFY ---- NORMAL ----  TIMEOUT/CANCEL  --- DEADLOCK ---
  PAGESET            TYPE      NAME                   SUSPENDS LATCH  IRLMQ  OTHER NMBR          AET NMBR          AET NMBR          AET
 ------------------- --------- ---------------------- -------- -----  -----  ----- ---- ---------- ---- ---------- ---- ----------
                     ROW       PAGE=X'00100106'            343   343      0      0  343    0.268938    0         N/C    0         N/C
                               ROW =X'25'                    0     0      0      0
                     ROW       PAGE=X'00100106'            124   123      0      0  124    0.154899    0         N/C    0         N/C
                               ROW =X'26'                    1     0      0
                     ROW       PAGE=X'00100106'            174   174      0      0  174    0.153716    0         N/C    0         N/C


)                    ** SUM OF GLT0             **        1291  1111      0      0 1291    0.157569    0         N/C    0         N/C
                                                            148     0     32
```

Figure 169: DB2PM LOCKING REPORT for GLT0

In this report, note that in 7 minutes elapsed time, there were 202 seconds (1291 * 0.15 seconds) reported delay. Almost all events (1111 or 1291 total) were row lock (LOCAL) contention. One could check ST04 “global times” to review the delay as a percentage of time in DB2.

Assuming that no change to the ledger definitions in GLT0 is possible then one would test to find the level of parallelism with the maximum total throughput.

9. Health Check

9.1. Check for SAP instance-level or system-level problems

9.1.1. Application server OS paging

When application server paging occurs, there are several possible actions to alleviate it:

- Check for jobs that are memory hogs, and ensure that they are efficiently coded. For example, a program may be building an internal table, where not all columns in the table are needed. ST02 can be used to find running jobs that use lots of memory: ST02 > drill into Extended Memory > press "mode list". STAT records also display memory usage.
- Reduce the number of work processes on the system
- Add more memory to the application server
- If you can't get rid of paging, manage workload and live with paging. As shown in section 9.2.1, one can calculate "page-ins per active CPU second": *(page-ins per second / processors / CPU utilization)*. If there is good performance on I/O to pagefiles (that is, there are several pagefiles on write-cached disk) one can probably live with some moderate paging. Configure the system to keep "page-ins per active CPU second" in the low single digits.

9.1.2. Application server CPU constraint

When there is a CPU constraint on the application server:

- Review program activity (STAT, STAD, ST03) on the system at peak CPU times to see if this might be a symptom of programs that are inefficient and using too much CPU. Check for programs that spend the vast majority of their time doing CPU processing on the application server – they may be coded inefficiently. Use SE30 to profile and analyze activity in the ABAP.
- If there is an unusually high amount of operating system kernel CPU time, then it could be an indication of a problem in the OS, or in the way that the SAP kernel calls OS kernel services. This is rare, but happens on occasion. Open an OSS message.
- Configure fewer work processes on the application server.
- Add more application servers.

9.1.3. SAP managed memory areas

When SAP managed areas, such as roll, generic buffer, program buffer, or EM are overloaded, it will impact SAP performance:

- If the roll memory area fills, then SAP will roll to disk on that application server, causing long roll-in and roll-out times.
- If the generic buffer fills, then tables which should be buffered in SAP memory will not be buffered. This increases the load on the DB server, and impacts performance of transactions that use these tables.
- If the program buffer fills, then programs must be loaded from the database server, which increases load time in SAP.

- If Extended Memory (EM) fills, all work processes on the instance will start to allocate memory out of work process private area. When a dialog step allocates memory from the work process private area, the dialog step cannot roll out until finished. This can cause "wait for work process" delays.
- It is less serious when an individual process expands past its EM quota. In this case, the individual dialog step is in PRIV mode, and cannot be rolled out until it is finished. If this happens with only a couple work processes, it will probably not impact the instance.

9.1.4. Table buffering

SAPnote 47239 describes the behavior of buffered tables. Tables can be buffered in individual rows in the single record buffer, which is accessed via "select single", or by sets of rows in the generic buffer, which is accessed by "select" or "select single".

When buffering a table in the generic buffer, take into account the way that the table is accessed. For instance, if the table is buffered in ranges based on the first three key columns, but is read in ranges based on the first two key columns, then the table cannot be read from buffer. The table must be accessed with as many or more columns as were specified in the generic buffering configuration.

There are four common problems related to buffering:

- A table is not buffered, but should be. If a table has a moderate size, is generally read only, and the application can tolerate small time intervals where the buffered copy is not the latest version, then the table is a candidate for buffering.
- Tables can be buffered in the wrong category. The ABAP "select" statement reads only from the generic buffer, but does not read from the single record buffer. The ABAP "select single" can read from the generic buffer or single record buffer. Use ST10 statistics to check whether select or select single is generally used with a table.
- The SAP buffer is too small to hold all the tables that should be buffered, as described in section 9.1.2
- A table is changed frequently, and should not be buffered. Buffered tables have to be re-synchronized when changed, which causes additional load on the DB server and application server, if the changes occur too frequently.

9.1.5. Wait time

This was discussed above in section 7.1.1:

- It can be a symptom of other problems. A performance problem that elongates the time required for the dialog step to finish (CPU overload on application server, DB server performance problems, roll-in and roll-out, etc) can cause all the work processes to fill up and then cause wait time.
- It can be problem itself, where too few work processes are configured.

9.1.6. Number ranges

SAP uses number ranges to create sequence numbers for documents. Number ranges can be:

- Not buffered, where each number range is a single row in the table NRIV. In this case, number range sequence numbers given to the application are sequential by time, and no numbers are lost.
- Configured with NRIV_LOKAL, where each application server or work process can have its own row in NRIV for the number range. In this case, sequence numbers may be out of time sequence, but no numbers are lost.
- Buffered in SAP. In this case, sequence numbers may be out of time sequence, and numbers may be lost.

The choice between the three is made based on business and legal requirements for document sequence numbers.

Number range buffering problems are often found during stress tests, or early after go-live:

- If the problem is contention for a non-buffered number range, the symptom is long “direct read” times on NRIV. NRIV is the table that contains number ranges, and it is read with “select for update”. Using SM50 or SM66, look for “direct read” NRIV.
- If the problem is contention on buffered row in NRIV, where buffering quantity is too small, the problem can also be seen in SM50 or SM66 where work processes are “stopped NUM” or waiting on semaphore 8 waiting for the buffered numbers to be replenished.

9.2. Sample SAP instance-level and system-problems

9.2.1. Application server paging

ST06 > detailed analysis > previous hours memory - display the screen where one can see historical statistics, to look for paging or CPU constraints.

Fri Mar 1 01:10:02 2002

Hour	Pages in /h	Pages out /h	Paged in [Kb/h]	Paged out [Kb/h]	Free memory/h in Kbyte minimum	maximum	average
1	2.094	24	8.376	96	0	5.288	1.586
0	6.540	1.303	26.160	5.212	0	90.980	1.919
23	0	0	0	0	560	182.316	2.628
22	4.330	51	17.320	204	940	1.383.060	12.895
21	6.249	44	24.996	176	48	132.804	2.430
20	0	0	0	0	1.160	246.192	4.328
19	18.533	77	74.132	308	280	221.868	3.109
18	23.194	129	92.776	516	0	362.572	6.126
17	0	0	0	0	992	380.776	16.415
16	200.669	648	802.676	2.592	0	636.016	24.655
15	0	0	0	0	0	833.312	56.550
14	134.325	5.776	537.300	23.104	0	496.932	12.843
13	363.431	576.967	1453.724	2307.868	0	564.740	6.894
12	0	0	0	0	0	590.168	8.124
11	238.975	401.581	955.900	1606.324	0	537.068	7.007
10	0	0	0	0	0	167.044	3.390
9	135.087	639.084	540.348	2556.336	0	461.952	6.407
8	38.674	2.799	154.696	11.196	0	535.060	11.049
7	0	0	0	0	0	324.556	24.918
6	2.391	64	9.564	256	1.204	375.264	29.181
5	0	0	0	0	608	294.524	9.099
4	21.764	270	87.056	1.080	0	181.244	2.519
3	0	0	0	0	0	129.872	3.082
2	0	0	0	0	852	293.236	3.021

Figure 170: ST06 > detail analysis > previous hours memory - high paging

In the above example, the peak page-in rate is 363,431 per hour. There were several periods with paging rates over 100K per second.

One can gather CPU utilization statistics for this period (not shown in this document) and use the information to calculate the “page-ins per active CPU second”. The system in this example was a 24-way. During the interval 13:00 to 14:00, the CPU utilization averaged 40%. (These screens are not included here.) *363,431 page in per hour / 3600 seconds per hour / 24 processors / 0 .40 utilization* = 10.5 “page ins per active CPU second”, which is too high, compared to our rule-of-thumb in section 9.1.1.

During the interval 14:00 to 15:00, the CPU utilization averaged 30%. *134,325 / 3600 / 24 / 0.30* = 5.1, which is at the high end of our ROT.

Since paging problems will impact all users on the system, with paging being moderate to high during 5 hours of the day (9, 11, 13, 14, 16), this problem should be addressed.

9.2.2. Application Server CPU constraint

This is a problem that is relatively simple to see, either via ST06(N) history or ST06(N) current statistics. In the ST06(N) main panel, there is a current utilization display. ST06 > "detail analysis" can be used to display hourly averages for the previous days. Figure 171 is the current utilization display.

```
Mon Jul  3 12:57:56 2000
 interval  10   sec.
CPU--------------------------------------------------------------------------
Utilization  user    %            98   Count                            24
             system  %             0   Load average     1 min        28.00
             idle    %             2                    5 min        28.80
System calls/s                3,257                   15 min        26.38
Interrupts/s                      0   Context switches/s            1,405


Memory-----------------------------------------------------------------------
Physical mem avail  Kb    16,777,116   Physical mem free   Kb    10,440,828
Pages in/s                        0   Kb paged in/s                      0
Pages out/s                       0   Kb paged out/s                     0


Swap-------------------------------------------------------------------------
Configured swap     Kb     9,371,648   Maximum swap-space  Kb     9,371,648
Free in swap-space  Kb     9,357,328   Actual swap-space   Kb     9,371,648


Disk with highest response time----------------------------------------------
Name                             cd0   Response time       ms             0
Utilization                        0   Queue                            N/A
Avg wait time       ms           N/A   Avg service time    ms             0
Kb transfered/s                    0   Operations/s                       0


Lan (sum)--------------------------------------------------------------------
Packets in/s                     632   Errors in/s                        0
Packets out/s                    636   Errors out/s                       0
Collisions                         0
```

Figure 171: ST06 CPU constraint

ST06 does not report CPU use correctly for AIX systems running with SPLPAR configuration. If the SAP application server is on an SPLPAR AIX system, then install SAP note 994025, and use ST06N.

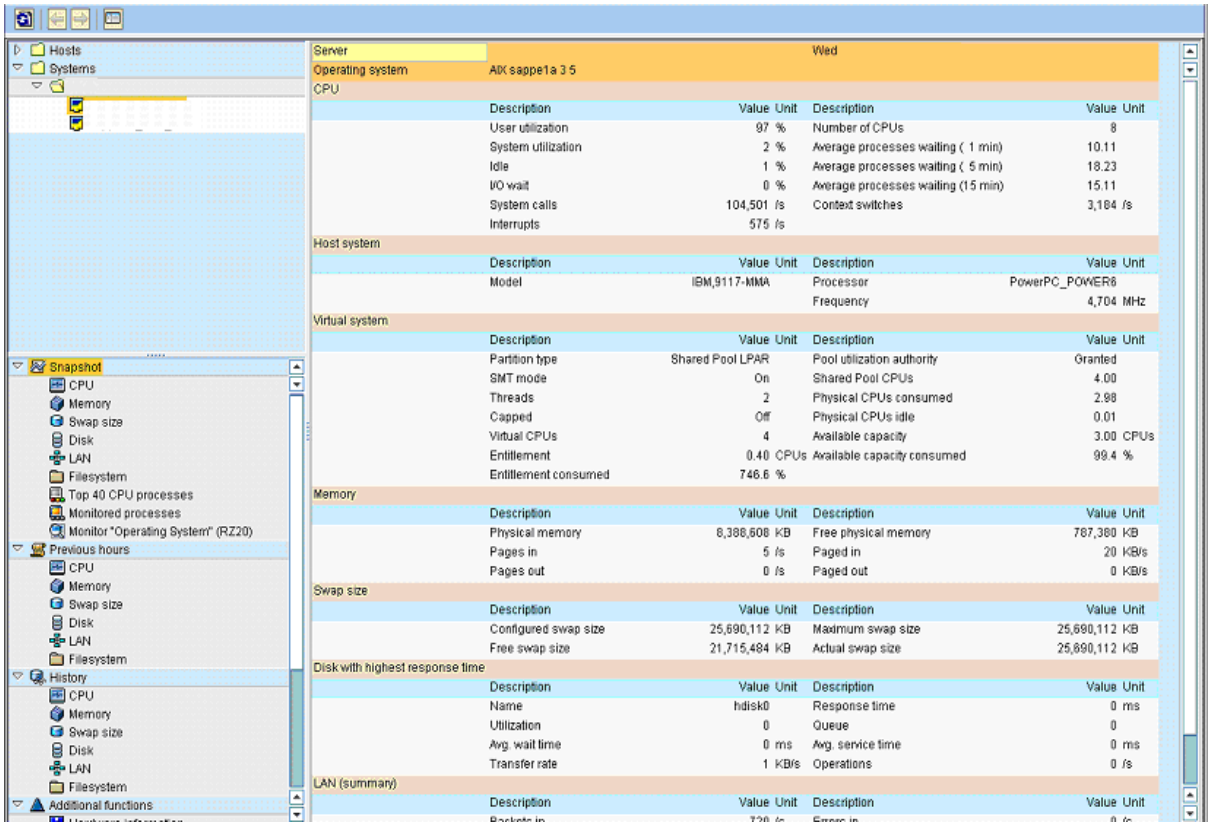


Figure 172: ST06N with SPLPAR statistics

As shown in Figure 172, the SPLPAR-aware ST06N can report on SPLPAR statistics such as shared pool use, entitled capacity, etc.

With ST06N, one can drill directly into the history for the CPU, memory, etc.

One can also display the hourly average CPU statistics. If hourly averages are high (over 75% or so) it is very likely that there are peak times of 100% utilization. Tools that report on smaller intervals than one hour (e.g. vmstat, iostat, sar) would show more detail on CPU utilization.

Hour	CPU utilization in %			Interrupts	System calls	Context switches
	User	System	Idle	/h	/h	/h
8	4	0	96	0	269.189	934.217
7	7	0	93	0	551.034	312.400
6	7	0	93	0	579.646	1.110.890
5	0	0	100	0	969.885	483.551
4	0	0	100	0	815.729	317.233
3	0	0	100	0	784.376	308.048
2	14	0	86	0	915.310	700.836
1	96	0	4	0	813.304	786.740
0	99	0	1	0	490.488	85.791
23	96	0	4	0	269.870	287.457
22	56	0	44	0	896.196	779.490
21	2	0	98	0	193.229	1.015.955
20	81	1	18	0	854.058	1.160.147
19	86	1	13	0	167.127	1.062.453
18	65	1	34	0	1.047.213	1.149.975
17	47	0	53	0	19.504	925.161
16	54	0	46	0	747.990	290.571
15	38	1	61	0	645.075	764.281
14	70	1	29	0	22.985	315.842
13	1	0	99	0	149.242	1.011.789
12	2	0	98	0	662.668	55.023
11	0	0	100	0	351.960	392.877
10	0	0	100	0	453.605	268.629
9	0	0	100	0	460.793	374.825

Figure 173: ST06 > detail analysis > previous hours CPU - CPU constraint

Since the hourly averages will not show short periods when the CPU usage goes to 100%, other monitoring tools would be needed to detect shorter periods of CPU constraint.

As mentioned earlier in Section 7.2.11 on missing time, the symptom in STAD of a CPU overload on the application server is that processing time is much larger than CPU time.

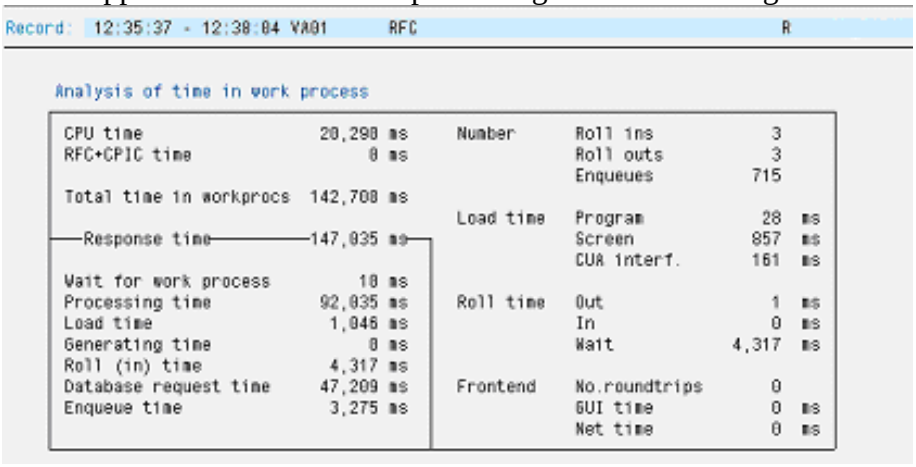


Figure 174: STAD from system with CPU overload

9.2.3. Roll Area shortage

When the roll area is too small, it will delay dialogs steps on roll-in and roll out. Note that in Figure 175 Roll area "Max use" is larger than "In memory". This shows that SAP has filled the memory roll area, and was rolling to disk. Roll area "current use" is larger than "in memory", which shows that SAP is rolling to disk at the time of this screen shot. ST03 and STAT/STAD will show information on the length of delay this causes for dialog steps. The SAP parameter *rdisp/ROLL_SHM* controls roll area memory allocation.

```
System                  : des0101_SB1_00          Tune summary
Date & time of snapshot: 02/01/2000  01:02:49     Startup: 01/31/2000  19:24:33
```

Buffer	Hitratio [%]	Allocated [kB]	Free space [kB]	[%]	Dir. size Entries
Nametab (NTAB)					
Table definition	97.90	5,043	4,048	98.68	30,000
Field description	94.45	32,348	29,305	97.68	60,001
Short NTAB	97.12	4,848	2,474	98.96	60,001
Initial records	98.93	6,348	3,855	96.38	60,001
Program	97.57	354,683	155,456	44.42	43,750
CUA	98.43	6,000	5,248	92.95	3,000
Screen	99.50	19,531	19,005	98.91	4,500
Calendar	100.00	488	400	83.68	200
Tables					
Generic key	99.99	341,797	330,556	98.62	45,000
Single record	83.77	30,000	29,260	97.73	500
Export/import	50.00	4,096	3,749	100.00	2,000

SAP memory	Current use [%]	[kB]	Max. use [kB]	In memory [kB]	On disk [kB]
Roll area	31.18	79,824	82,016	64,000	192,000
Paging area	0.02	40	64	8,000	248,000
Extended Memory	0.90	105,472	107,520	11750,400	
Heap Memory		0	0		

Figure 175: ST02 roll area over-committed

When monitoring a running system that has a roll-area shortage, you would see many processes waiting on the action “roll in” and “roll out”. In this SM66 display, the central instance has run out of roll area, and the other instances that make CPIC or enqueue calls to the CI are delayed in turn.

Sort: Server												
Server	No	Typ	PID	Status	Reaso	Se	Start	Err	CPU	Time	Cli	User	Report	Action/Reason for waiting
des0101_SB1_00	0	DIA	58114	running		Yes			1	010	XXX_BATCH			Roll Out
des0101_SB1_00	1	DIA	58482	running		Yes				010	XXX_BATCH			Roll Out
des0101_SB1_00	2	DIA	52776	running		Yes			1	010	XXX_BATCH			Roll In
des0101_SB1_00	3	DIA	51198	running		Yes			1	010	XXX_BATCH			Roll Out
des0101_SB1_00	4	DIA	50934	running		Yes				010	XXX_BATCH			Roll Out
des0101_SB1_00	5	DIA	50122	running		Yes			1	010	XXX_BATCH			Roll Out
des0101_SB1_00	6	DIA	49708	running		Yes			1	010	XXX_BATCH			Roll Out
des0101_SB1_00	7	DIA	49570	running		Yes				010	XXX_BATCH			Roll In
des0101_SB1_00	8	DIA	43922	running		Yes				010	XXX_BATCH			Roll Out
des0101_SB1_00	9	DIA	17738	running		Yes				010	XXX_BATCH			
des0101_SB1_00	10	DIA	17458	running		Yes				010	XXX_BATCH			Roll In
des0101_SB1_00	11	DIA	15432	running		Yes				010	XXX_BATCH			Roll Out
des0101_SB1_00	12	DIA	14242	running		Yes				010	XXX_BATCH			Roll In
des0101_SB1_00	14	DIA	54406	running		Yes				010	XXX_BATCH			Roll Out
des0101_SB1_00	15	DIA	28654	running		Yes				010	XXX_BATCH			Roll Out
des0101_SB1_00	16	ENQ	12594	running		Yes								
des0101_SB1_00	19	DIA	43188	running		Yes				010	XXX_BATCH			Roll In
des0797_SB1_00	23	BTC	28776	running		Yes			383	010	XXX_BATCH	SAPLPGRW		
des0797_SB1_00	24	BTC	26936	stopped	ENQ	Yes			143	010	XXX_BATCH	SAPLSENA		
des0797_SB1_00	25	BTC	28578	stopped	CPIC	Yes			469	010	XXX_BATCH	SAPLSENT	CMSEND(SAP) / 85179598	
des0797_SB1_00	26	BTC	26556	stopped	CPIC	Yes			586	010	XXX_BATCH	SAPLSENT	CMSEND(SAP) / 83211579	
des0797_SB1_00	27	BTC	25804	stopped	ENQ	Yes			391	010	XXX_BATCH	SAPLSENA		
des0797_SB1_00	28	BTC	25592	stopped	CPIC	Yes			115	010	XXX_BATCH	SAPLSENT	CMSEND(SAP) / 82180132	
des0797_SB1_00	29	BTC	25270	stopped	CPIC	Yes			623	010	XXX_BATCH	SAPLSENT	CMSEND(SAP) / 82521911	
des08b3_SB1_00	11	BTC	36560	running		Yes			73	010	XXX_BATCH			Sequential read
des08b3_SB1_00	23	BTC	23718	stopped	ENQ	Yes			620	010	XXX_BATCH			

Figure 176: SM66 roll in and roll out

When SAP is rolling to disk, ST06 will show high I/O activity to the disk with the roll area. SAP parameter DIR_ROLL specifies the directory where the disk roll file is located.

```
Tue Feb  1 01:13:16 2000
 interval  10   sec.
 Disk        Resp.  Util.  Queue  Wait  Serv      Kbyte     Oper.
               [ms]    [%]   Len.  [ms]  [ms]       [/s]      [/s]
-------------------------------------------------------------------
|   hdisk1 |   1  |   3  | N/A | N/A |   1  |        1  |     3  |
|   hdisk4 |   1  |   0  | N/A | N/A |   1  |        0  |     0  |
|   hdisk5 |   1  |   0  | N/A | N/A |   1  |        0  |     0  |
|   hdisk6 |   1  |   0  | N/A | N/A |   1  |        0  |     0  |
|   hdisk9 |   1  |   0  | N/A | N/A |   1  |        0  |     0  |
|   hdisk0 |   0  |   3  | N/A | N/A |   0  |        1  |     3  |
|  hdisk10 |   0  |   0  | N/A | N/A |   0  |        0  |     0  |
|  hdisk11 |   0  |   0  | N/A | N/A |   0  |        0  |     0  |
|   hdisk2 |   0  |100   | N/A | N/A |   0  |       52  |   105  |
|   hdisk3 |   0  |   0  | N/A | N/A |   0  |        0  |     0  |
|   hdisk7 |   0  |   0  | N/A | N/A |   0  |        0  |     0  |
|   hdisk8 |   0  |   0  | N/A | N/A |   0  |        0  |     0  |
-------------------------------------------------------------------
```

Figure 177: ST06 > detail analysis > disk - high I/O activity on ROLL area

9.2.4. ST02 buffer area shortage

In this example, the generic area is too small for all the buffer-able tables to be buffered. This causes swaps, and extra database requests. This kind of problem will generally have a much greater impact on a specific program or programs (the ones which use the table which does not fit in buffer) than on the system overall.

The “Database accesses” counter is a better indicator than “swaps” for this problem. If a table cannot be buffered due to undersized buffers on the application server, it may cause thousands of unneeded database calls.

Here, generic key has millions of DB calls.

Buffer	Hitratio [%]	Allocated [kB]	Free space [kB]	Free space [%]	Dir. size Entries	Free directory Entries	Free directory [%]	Swaps	Database accesses
Nametab (NTAB)									
Table definition	99.91	19,191	10,676	72.24	80,500	58,148	72.23	0	01,717
Field description	99.99	94,289	23,908	27.17	80,500	58,996	73.29	0	24,453
Short NTAB	99.98	6,016	2,374	67.83	20,125	15,439	76.72	0	4,710
Initial records	99.99	13,516	6,370	57.91	20,125	2,741	13.62	0	17,461
Program	99.97	1,000,000	5,320	0.56	249,999	224,768	89.91	7,964	67,854
CUA	99.98	20,000	2,773	15.85	10,000	6,326	63.26	0	4,004
Screen	99.91	48,242	8,840	18.78	20,000	17,257	86.29	12,469	16,677
Calendar	100.00	635	428	69.26	300	89	29.67	0	211
OTR	96.68	4,096	3,570	99.42	2,000	1,916	95.80	0	0
Tables									
Generic key	99.83	101,563	5,518	5.63	15,000	4,849	32.33	1,976	5,460,626
Single record	99.65	30,000	3,083	10.35	500	325	65.00	11	580,397
Export/import	93.28	100,000	19,494	22.34	51,000	0	0.00	359,665	0
Exp./Imp. SHM	97.19	4,096	3,558	99.08	2,000	1,996	99.80	0	0

Figure 178: Generic buffer swapping

One can look at this problem in more detail by drilling into the “generic key” line in ST02, and then selecting “buffered objects”. This will display the state of tables in the buffer, and the count of calls for each table. Last, sort by calls or “rows affected” to pull the problems to the top of the list.

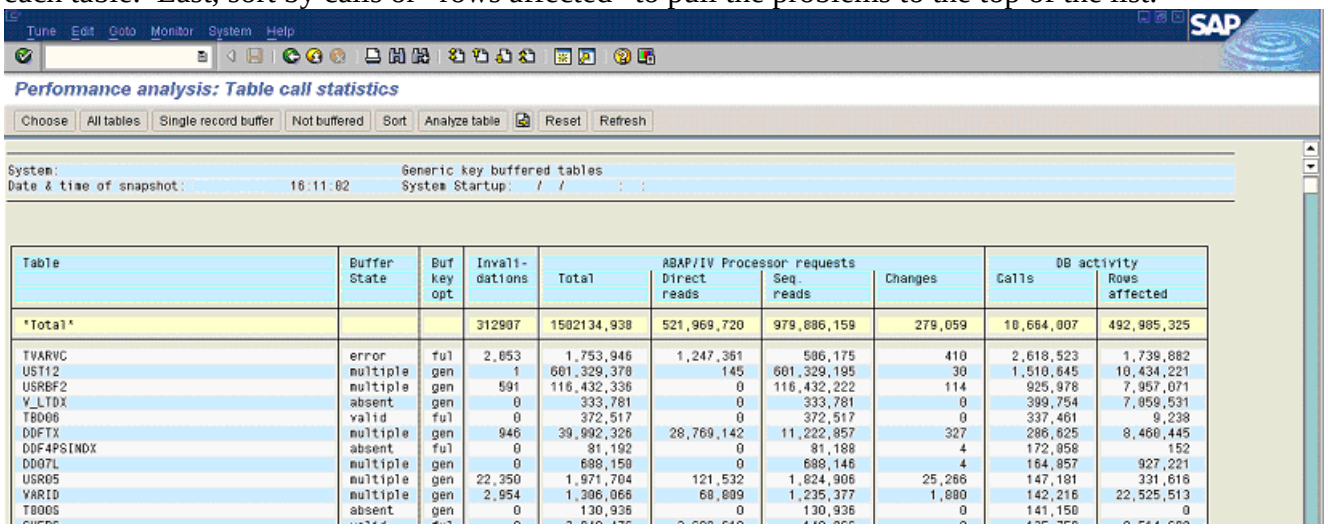


Figure 179: ST02 > drill into generic key > buffered objects

After sorting the list by calls we see a number of tables with the state “error”, which generally means that the table would not fit into the buffer. See SAP Note 3501 for information on how to interpret the state of tables in ST02.

In Figure 179, drill into a line to see the details on a table, such as how much space it needs to be buffered.

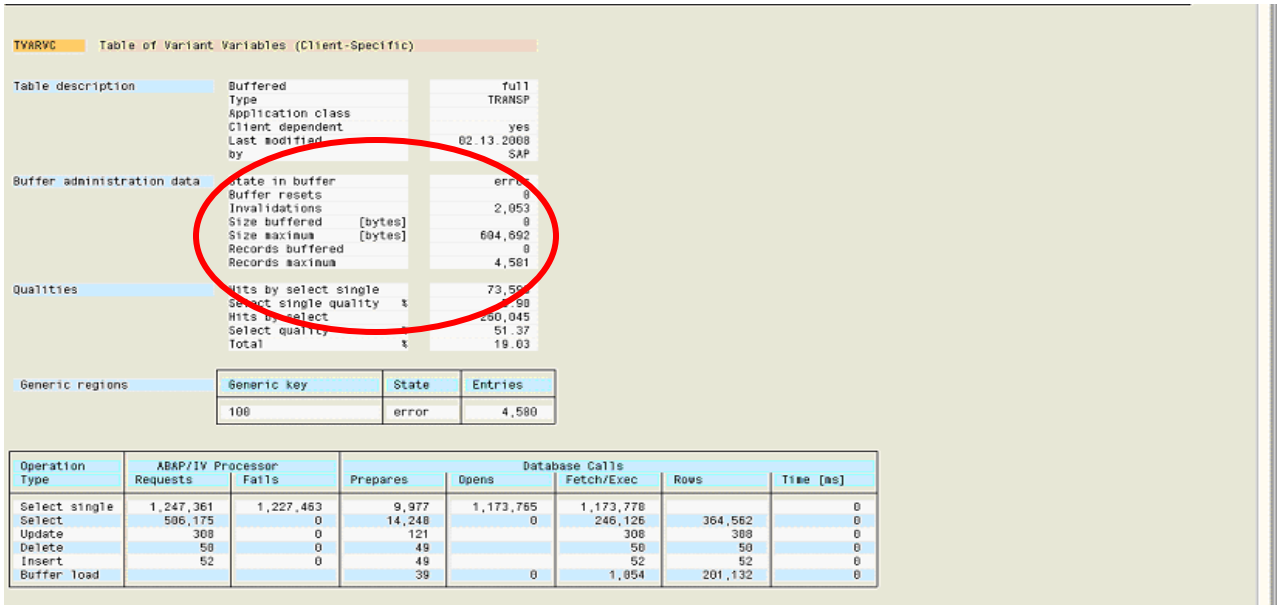


Figure 180: ST02 table details

9.2.5. Find table buffering candidates

Check ST10 “not buffered” tables, to see if there are any candidates for buffering. Sort the ST10 list by DB calls. Use weekly ST10 statistics, to average out the impact of daily differences.

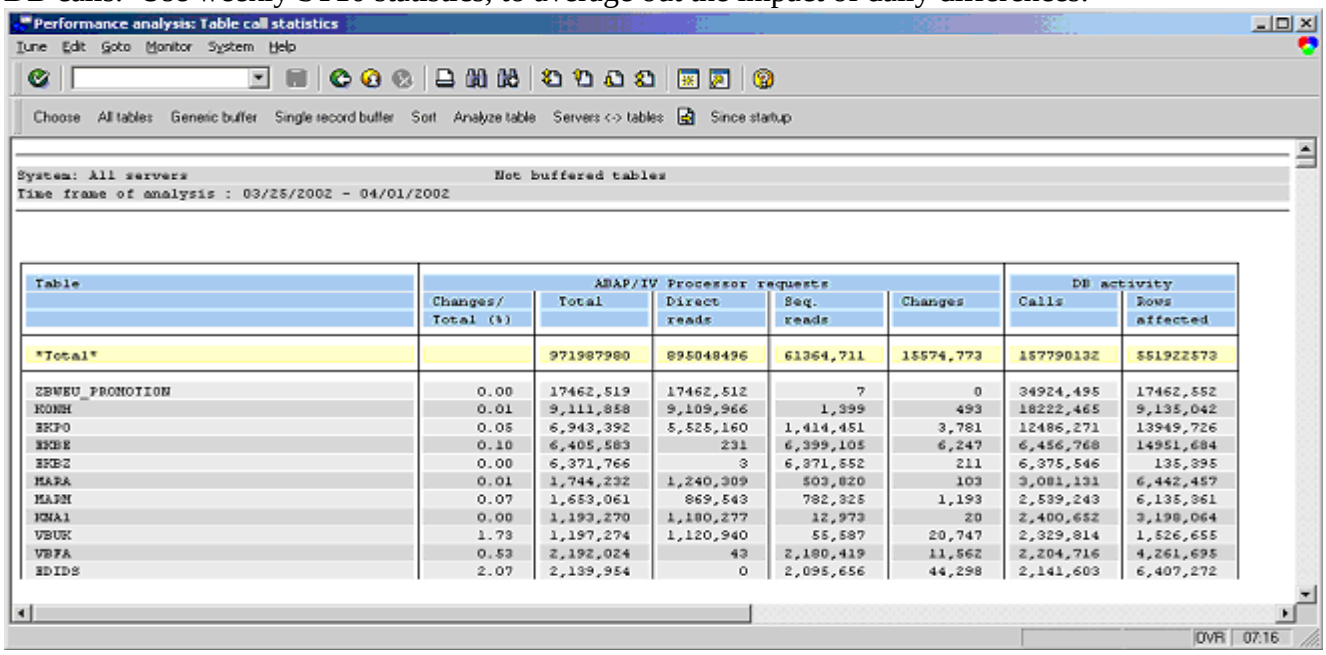


Figure 181: ST10 > not buffered, previous week, all servers > sort by calls

Here, note that the top entry has no changes. “Read only (for the most part)” is our first criteria for a candidate.

Now, check how large the table is. One can use SE16.

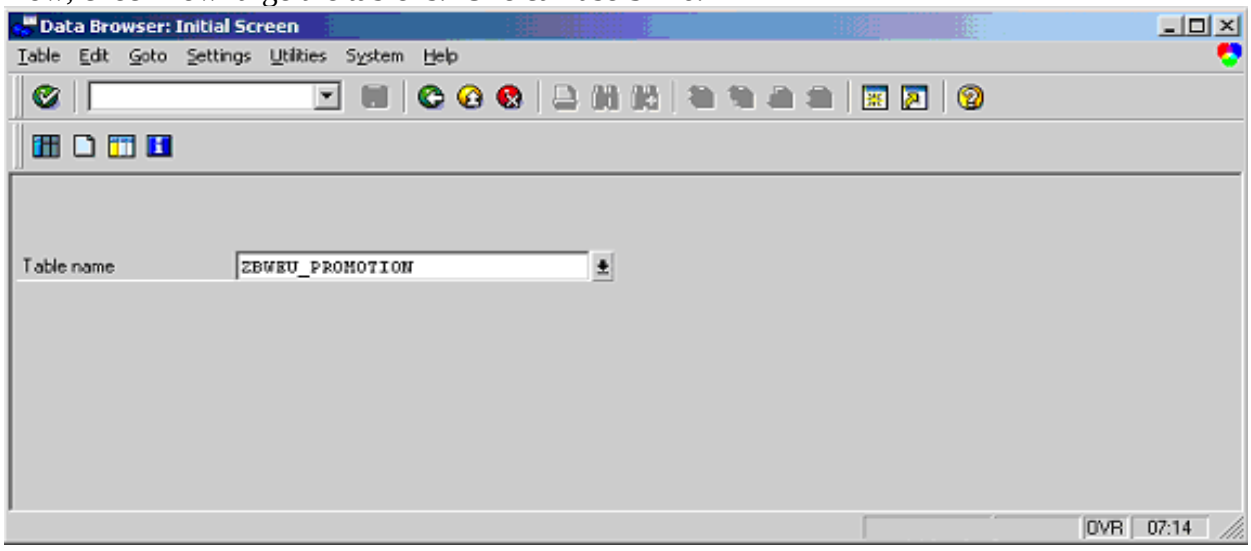


Figure 182: SE16 for counts

Enter the tablename, press execute, then press “number of entries”. This information can also be found more efficiently by checking catalog statistics with DB02, or ST04 DB2 catalog browser. They read the catalog statistics, rather than counting the rows in the table.

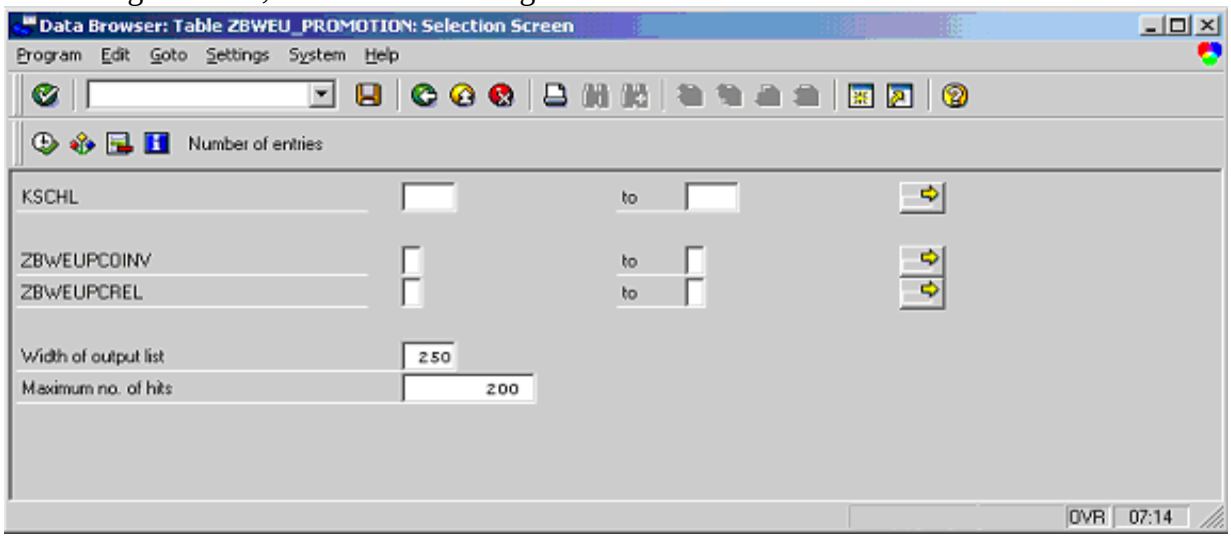


Figure 183: SE16 counts part two

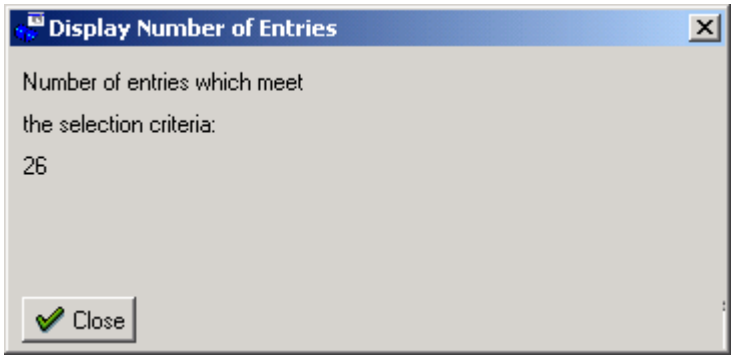


Figure 184: SE16 count result

There are 26 rows, so it meets the second criteria, moderate size.

To evaluate whether the application can tolerate small time intervals when the buffered data is out of synch with the database, contact an application expert for this area. Use the name of the creator of the table (in SE11) as a starting point to find the expert.

Since the table is usually read by select single, it can be buffered in single record buffer. Since select single can read from the generic buffer, and the table is very small, it can also be put in generic buffer, fully buffered.

The three tests for buffering were:

- Read only (for the most part)
- Moderate size (up to a few MB, larger for very critical tables)

- Application can tolerate data being inconsistent for short periods of time

9.2.6. Table buffered with wrong attributes

In this case, examine the tables buffered in single record buffer, to search for tables that are generally read with select. ABAL "select" does not read from the single record buffer, so tables set this way will not be read from buffer.

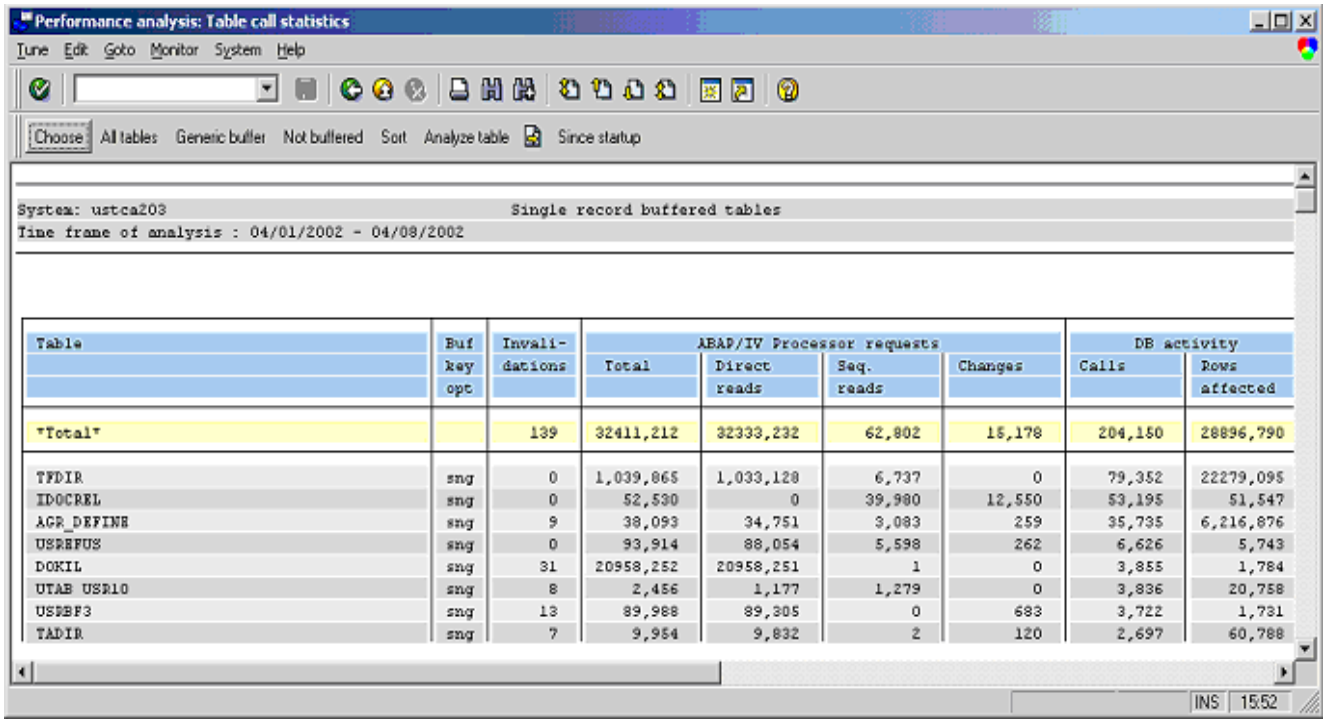


Figure 185: single record buffering on table accessed via select

In Figure 185 the table that is second by calls, IDOCREL, is single record buffered, and only read with select. Since select does not access the single record buffer, the buffering in ineffective. (Actually, IDOCREL has another problem too – it is frequently changed. It should not be buffered.)

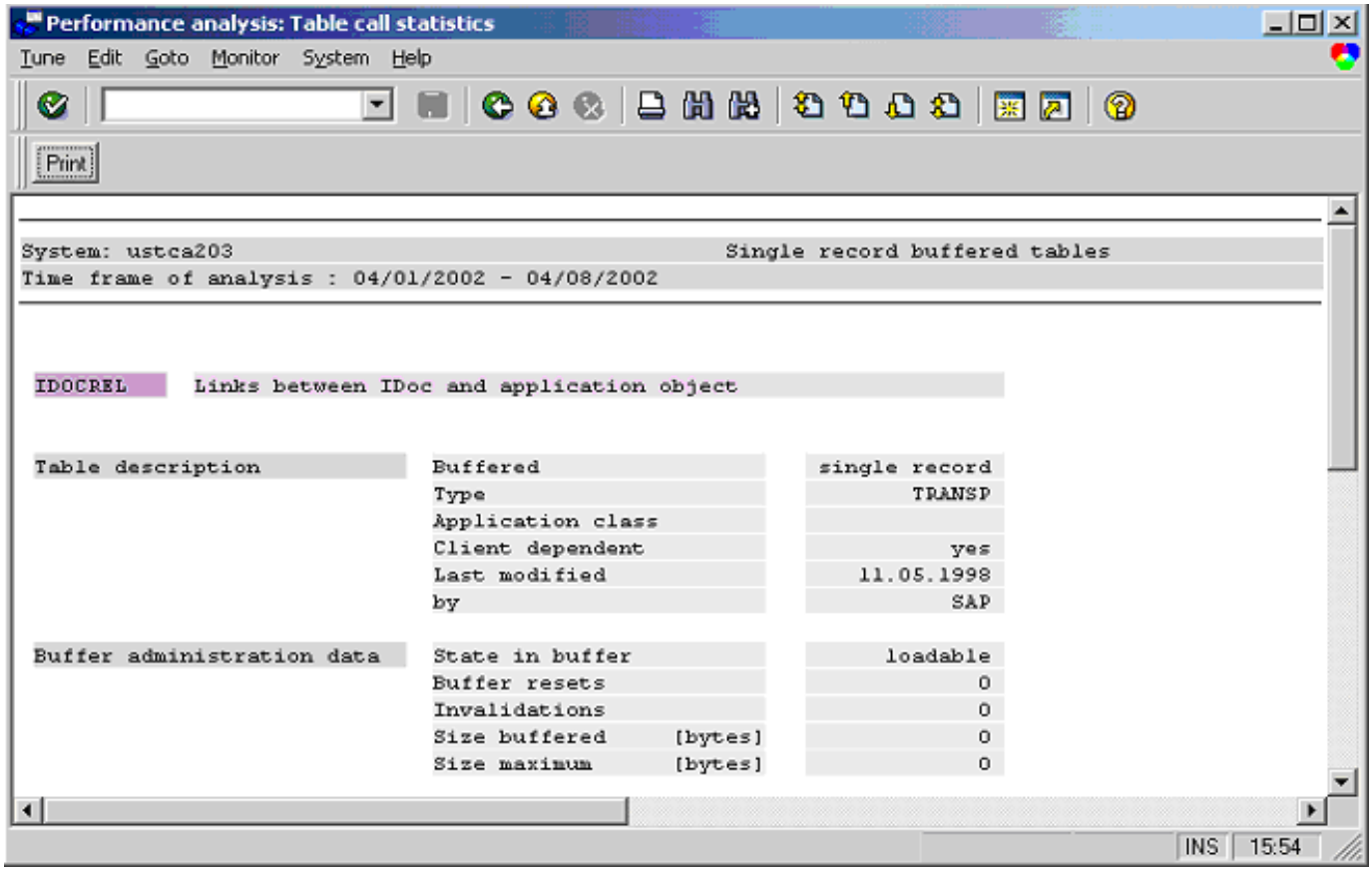


Figure 186: ST10 table details

Drill into the table in ST10, and we see that it is not present in the buffer (state is loadable). If we want this table to be buffered, the "technical settings" would need to be set to generic buffering, if an appropriate generic key could be determined, and if it were changed less frequently.

Since this is an SAP table, we would check SAP Notes, and open an OSS message regarding changing the technical settings.

9.2.7. Number range buffered by quantity that is too small

When reviewing a running system with SM50, note processes “stopped NUM” and waiting for semaphore 8. These are symptoms of a performance problem while getting a sequence number from a number range. If the number range was buffered with a buffer quantity that is too small, SAP has to replenish the buffered number range frequently, and this causes lock contention in the database for rows in the table NRIV. (The quantity of sequence numbers in a buffered set is an option in SNRO.)

No.	Ty.	PID	Status	Reason	Start	Err	Sem	CPU	Time	Program	User	Action	Table
0	DIA	70632	running		Yes					<no buff	000 SAPSYS	Direct read	NRIV
1	DIA	64766	waiting		Yes								
2	DIA	60848	waiting		Yes								
3	DIA	60564	waiting		Yes								
4	DIA	59182	waiting		Yes								
5	DIA	55944	running		Yes					RSMON000	440 IOC4779		
6	DIA	55638	waiting		Yes								
7	DIA	54966	stopped	CPIC	Yes				1	SAPMSSY1	440 T060504		
8	DIA	54300	waiting		Yes								
9	DIA	53246	waiting		Yes								
10	DIA	52622	waiting		Yes								
11	DIA	52392	stopped	CPIC	Yes				2	SAPMSSY1	440 T060504		
12	UPD	49918	running		Yes					SAPLBOIF	440 T060504	Sequential read	VBDATA
13	UPD	47376	running		Yes					RSM13000	440 T060504	Sequential read	VBMOD
14	UPD	46500	running		Yes				1	RSM13000	440 T060504	Commit	
15	UPD	46186	running		Yes					RSM13000	440 T060504	Commit	
16	BTC	45558	running		Yes				5117	SAPLARFC	440 T060504	Insert	ARFCSSTATE
17	BTC	40806	stopped	NUM	Yes				967	SAPLSNR3	440 T060504		
18	BTC	39320	running		Yes		8		1	SAPLSNR3	440 T060504		
19	BTC	38332	running		Yes		8		5028	SAPLSNR3	440 T060504		
20	BTC	38066	running		Yes				2162	SAPLSZA0	440 T060504	Insert	ADRT
21	BTC	37794	running		Yes				5729	SAPLIL00	440 T060504	Insert	IFLOT
22	BTC	37408	running		Yes				6	SAPLBUD1	440 T060504		
23	BTC	36430	running		Yes				791	SAPLES05	440 T060504	Direct read	KNA1
24	SPO	80132	waiting		Yes								
25	UP2	79242	running		Yes				1	SAPLSCD0	440 T060504	Insert	CDCLS
26	UP2	73914	running		Yes					RSM13000	440 T060504	Delete	VBHDR

Figure 187: SM50 “stopped NUM” and Sem 8

This problem can also be seen in the statement cache -- note the long “select for update” times on the table NRIV. Here the average elapsed time (old format of ST04 statement cache – see second column from right) for each select is 36 ms, which is rather long. Normally one might expect a few ms, at most.

Executions	Rows examined	Rows processed	Accumulated elap. time	Rows exam. / execs	Rows proc. / execs	Rows proc. / examined	Elap. time / execs	Statement text
79092	158182	79091	48:21.475709	2.000	1.000	0.500	36.685	SELECT * FROM "NRIV"

Figure 188: NRIV row-lock contention on 4.0 ST04 statement cache

With the 4.6 format statement cache (a screen shot from a different system than the rest of this example), look at the Timers tab in ST04 statement cache statistics, and sort by “elapsed time”. See the *last row* in Figure 189, where NRIV updates take 56 ms, on the average.

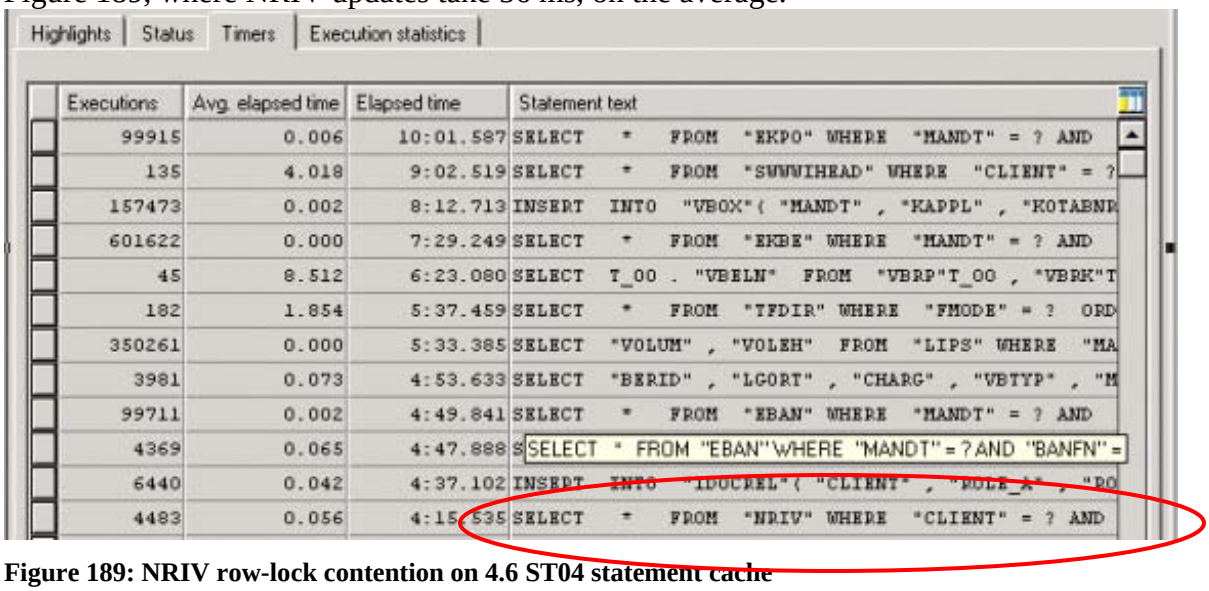


Figure 189: NRIV row-lock contention on 4.6 ST04 statement cache

As confirmation of the problem, in ST02 (detail analysis > number ranges> statistics) look at the statistics of number range performance, broken down into the time it takes a program to get a buffered number range (buffer time), and the time the <no buffer> server takes to get a new set of numbers when all the numbers buffered in memory have been given out (server time). Note that most of the times to replenish the buffered number ranges (server times) are very high. It often takes over 100 ms to update a single row in NRIV. When number range performance is good, the counters should be clustered toward the left of both “buffer times” and “server times”.

Number range buffer statistics

System:	sap019_QA5_48	
Date & Time of Snapshot:	23.07.1999	17:57:03
Startup:	23.07.1999	14:41:14

Number range buffers overview

Max. number of entries	1,000
Curr. number of entries	12
Max. number of indexes	3,000
Curr. number of indexes	12
Buffer size [bytes]	276,144
Buffer requests	234,196
Get requests	234,196
Server calls	12,491
Database requests	12,491
Collisions	0
Timeouts	0

Buffer times

<50us	<100us	<200us	<500us	<1ms	<2ms	<5ms	<10ms	<20ms	>=20ms
29237	47470	1270	489	198	260	691	1200	2518	38370

Server times

<1ms	<5ms	<10ms	<20ms	<50ms	<100ms	<200ms	<500ms	<1s	>=1ms
0	0	0	14	1255	3080	5033	2996	105	8

Figure 190: ST02 > detail analysis > number ranges - number range statistics

Another way to see the impact of this problem is with ST02 (detail analysis > semaphores). In order to interpret these statistics, take note of the time when the collection started, to calculate the delay time over the interval. See SAP note 33873 for information on this display. *Remember to turn the semaphore statistics off (settings > monitoring off) after viewing.*

Semaphore statistics Date 23.07.1999 Time 17.48.23 Monitoring Options: { SUM }

Wait time entering the critical path
Residence period in the critical path

Key	No.	<100us	<1ms	<10ms	<100ms	<1s	<10s	>10s	Time/ms	Protected area
1	16,811	15,618	708	482	3	0	0	0	1024	ABAP/4 program buffer
1	16,811	14,644	1,133	1,033	1	0	0	0	2048	
2	4,066	4,038	28	0	0	0	0	0	0	Work process communication
2	4,066	4,063	2	1	0	0	0	0	0	
3	2,342	2,322	18	2	0	0	0	0	0	APPC communication
3	2,342	2,338	4	0	0	0	0	0	0	
4	488	486	2	0	0	0	0	0	0	Terminal communication
4	488	487	1	0	0	0	0	0	0	
5	496	491	5	0	0	0	0	0	0	Work process communication
5	496	496	0	0	0	0	0	0	0	
6	282	213	8	56	5	0	0	0	0	Roll administration
6	282	0	0	282	0	0	0	0	1026	
7	14	14	0	0	0	0	0	0	0	Paging administration
7	14	13	0	1	0	0	0	0	0	
8	1,789	1,308	6	17	176	282	0	0	65s	Number range buffer
8	1,789	1,691	0	0	9	89	0	0	19s	

Figure 191: ST02 >detail analysis > semaphores - semaphore statistics

Since the number range was already buffered, we need to find the number range causing the problem, and increase its buffered quantity. Run a ST05 trace (selecting only NRIV table) to determine the number range which is causing the problem, then go to SNRO and increase the size of the buffered set size for this number range.

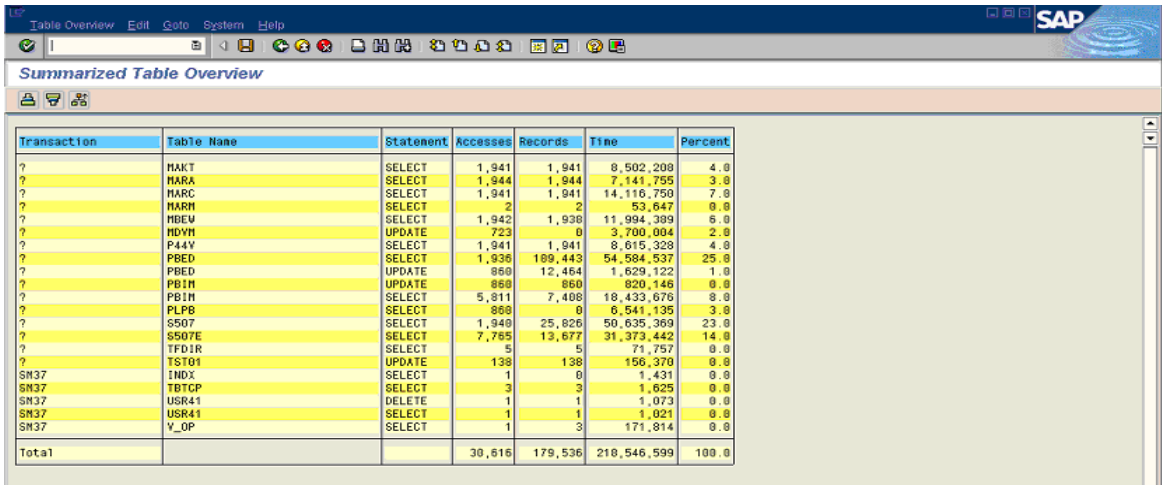
9.3. Check for network performance problems

The SAP planning guides for DB2 have information regarding configuration of network parameters for the SAP application. SAP note 1263782 has parameter settings for hipersockets networks.

9.3.1. Lost packets indicators

High DB request time can be caused by network problems such as dropped or lost packets. ST03 or STAT/STAD might point to this problem, if they show slow database request times while the internal database server indicators (ST04 times, ST04 statement cache elapsed times, etc) show good performance.

The most distinctive characteristic of lost packet problems is that the average SQL times are very bad, but when one runs an ST05 trace, most calls have good response times, with an occasional very long call.

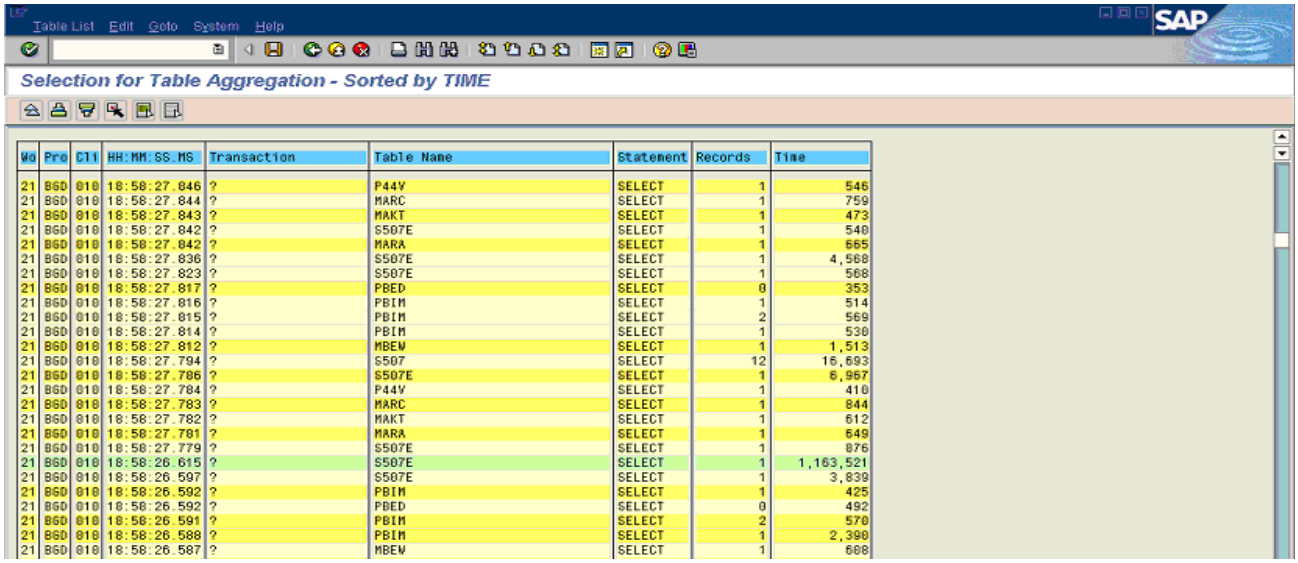


Transaction	Table Name	Statement	Accesses	Records	Time	Percent
?	MAKT	SELECT	1,941	1,941	8,502,208	4.0
?	MARA	SELECT	1,944	1,944	7,141,755	3.0
?	MARC	SELECT	1,941	1,941	14,116,750	7.0
?	MARM	SELECT	2	2	53,647	0.0
?	MBEW	SELECT	1,942	1,938	11,994,389	6.0
?	MDYN	UPDATE	723	0	3,700,004	2.0
?	P44V	SELECT	1,941	1,941	8,615,328	4.0
?	PBED	SELECT	1,936	189,443	54,584,537	25.0
?	PBED	UPDATE	860	12,464	1,629,122	1.0
?	PBIM	UPDATE	860	860	820,146	0.0
?	PBIM	SELECT	5,811	7,408	18,433,676	8.0
?	PLPB	SELECT	860	0	6,541,135	3.0
?	S507	SELECT	1,940	25,826	50,635,369	23.0
?	S507E	SELECT	7,765	13,677	31,373,442	14.0
?	TFDIR	SELECT	5	5	71,757	0.0
?	TST01	UPDATE	138	138	156,370	0.0
SM37	INDX	SELECT	1	0	1,431	0.0
SM37	TBTCP	SELECT	3	3	1,625	0.0
SM37	USR41	DELETE	1	1	1,073	0.0
SM37	USR41	SELECT	1	1	1,021	0.0
SM37	V_OP	SELECT	1	3	171,814	0.0
Total			30,616	179,536	218,546,599	100.0

Figure 192: ST05 summary with long average times

In Figure 192, average DB request times range from 5 ms to over 20 ms. This is very slow.

When we look at the un-summarized SQL trace, we see that most calls are very fast, but there is an occasional call that takes over one second.



Wo	Pro	Cli	HH:MM:SS.MS	Transaction	Table Name	Statement	Records	Time
21	BGD	010	18:58:27.846	?	P44V	SELECT	1	546
21	BGD	010	18:58:27.844	?	MARC	SELECT	1	759
21	BGD	010	18:58:27.843	?	MAKT	SELECT	1	473
21	BGD	010	18:58:27.842	?	S507E	SELECT	1	540
21	BGD	010	18:58:27.842	?	MARA	SELECT	1	665
21	BGD	010	18:58:27.836	?	S507E	SELECT	1	4,568
21	BGD	010	18:58:27.823	?	S507E	SELECT	1	568
21	BGD	010	18:58:27.817	?	PBED	SELECT	0	353
21	BGD	010	18:58:27.816	?	PBIM	SELECT	1	514
21	BGD	010	18:58:27.815	?	PBIM	SELECT	2	569
21	BGD	010	18:58:27.814	?	PBIM	SELECT	1	530
21	BGD	010	18:58:27.812	?	MBEW	SELECT	1	1,513
21	BGD	010	18:58:27.794	?	S507	SELECT	12	16,693
21	BGD	010	18:58:27.786	?	S507E	SELECT	1	6,957
21	BGD	010	18:58:27.784	?	P44V	SELECT	1	410
21	BGD	010	18:58:27.783	?	MARC	SELECT	1	844
21	BGD	010	18:58:27.782	?	MAKT	SELECT	1	612
21	BGD	010	18:58:27.781	?	MARA	SELECT	1	649
21	BGD	010	18:58:27.779	?	S507E	SELECT	1	876
21	BGD	010	18:58:26.615	?	S507E	SELECT	1	1,163,521
21	BGD	010	18:58:26.597	?	S507E	SELECT	1	3,839
21	BGD	010	18:58:26.592	?	PBIM	SELECT	1	425
21	BGD	010	18:58:26.592	?	PBED	SELECT	0	492
21	BGD	010	18:58:26.591	?	PBIM	SELECT	2	570
21	BGD	010	18:58:26.588	?	PBIM	SELECT	1	2,390
21	BGD	010	18:58:26.587	?	MBEW	SELECT	1	608

Figure 193: ST05 trace with symptom of network problem

While the SAP indicators can hint at a problem with lost packets, the problem must be pinpointed with OS and network tools. Switch and router settings, hardware problems, and operating system parameters can each cause lost packet problems.

9.3.2. Slow network indicators

The time required to make an SQL call from application server to database server varies with the type of network and network adapters used. Viewed with ST05, Escon based networks will

generally have median SQL call times of 4-8 ms, OSA-2 based networks (FDDI and Fast Ethernet) generally 3-6 ms, OSA Express Gigabit Ethernet generally 1-2ms. Hipersockets connections have call times of 400-800 microseconds. Networks, or network adapters, that are overloaded will not be able to achieve these times. Problems with overloaded OSA Express Gigabit Ethernet adapters are rare, due to the very high capacity of OSA Express.

A simple test of network performance is to run an ST05 SQL trace, summarize (ST05 > list trace > goto >summary), sort by time, and pull the slider bar on the right to the middle. If the median time is much higher than you would expect for your type of network, and the DB2 internal performance indicators (ST04 times, ST04 statement cache) look good, then there may be a network performance problem. Examine the CPU and paging activity on the DB server, to confirm that neither CPU overload nor paging is the cause of the consistently slow SQL. If the CPU and paging are OK, it increases the odds that it is a network problem.

For more detailed information on network performance, with a slight system overhead from tracing, the ST04 “ICLI trace” has an option to collect “network statistics”. This computes average time in the network. This trace subtracts the DB2 time from the SQL request time, and can give a very accurate indication of network performance when the DB server has a CPU or paging constraint. Please keep in mind that “network statistics” includes time on the physical network as well as the TCP/IP protocol stack on both application server and DB server, so system-wide problems such as CPU overload, paging, errors in workload prioritization can create long “network” times in the “network statistics”.

9.4. Sample network performance problems

9.4.1. Slow network performance example

The example is an examination of the performance of APO CIF. Start by examining the STAT records, and note that all the DB calls seem slow. Average update, insert, and deletes times are over 15 ms per DB call, which is slow.

Analysis of ABAP/4 database requests (only explicitly by application)

Database requests total	550	Request time	10,064 ms
		Matchcode time.	0 ms
		Commit time	147 ms
Requests on T??? tables	0	Request time	0 ms

Type of ABAP/4 request	Requests	Database rows	Requests to buffer	Database calls	Request time(ms)	Avg.time per req.
Total	550	35,064	65	416	10,064	18.3
Direct read	258	63	63		1,058	4.1
Sequential read	247	34,955	2	370	8,178	33.1
Update	43	43		43	646	15.0
Delete	1	2		2	16	16.0
Insert	1	1		1	19	19.0

Figure 194: STAT record with slow change SQL

Since all calls to the DB server are slow, there are several possible causes –

- CPU overload on DB server
- Incorrect configuration of priorities in WLM on DB server
- Network capacity or configuration problems
- Incorrect TCP/IP routing configuration
- Etc.

Use the ST04 ICLI "network statistics" trace, to determine the network time for database requests. The ICLI network statistics remove the time in DB2 from the request time for each database request sent from the SAP application server. What is left is time spent in the network protocol stack on application server and DB server, and time moving data across the network.

Be sure to turn the ICLI "network statistics" off after gathering data. Gathering these statistics places an additional load on the DB server and application server.

The ICLI "network statistics" are saved in the developer trace files, which can be viewed by AL11, ST11 or SM50.

```
ICLC2526I Network Statistics about sent packets (requests):
#PAC    #REC   AVG(KB)   MIN(KB)   MAX(KB)   | <128  | <256  | <512  | <1024 | <2048 | <4096 | <8192 | <16384| <31999| <64000
-------------------------------------------------------------------------------------------------------------------------------
2       1000     0.186     0.011     3.656        585     194     208       0       0      13       0       0       0       0
 [dbsldb2.c    2224]
C  *** ERROR =>
ICLC2527I Network Statistics about received packets (responses):
#PAC    #REC   AVG(KB)   MIN(KB)   MAX(KB)   | <128  | <256  | <512  | <1024 | <2048 | <4096 | <8192 | <16384| <31999| <64000
-------------------------------------------------------------------------------------------------------------------------------
2       1000     0.057     0.016     0.091       1000       0       0       0       0       0       0       0       0       0
 [dbsldb2.c    2224]
C  *** ERROR =>
ICLC2528I Network Statistics about time of request/response pairs:
#PAC    #REC   AVG(ms)   MIN(ms)   MAX(ms)   | <2    | <4    | <8    | <16   | <32   | >32
-----------------------------------------------------------------------------------------
2       1000    17.553     1.806   197.825          2     165     197     204     301     131
```

Figure 195: ST11 > display - ICLI network statistics in developer trace

In Figure 195, there are two things to check. First the average time on the network for each database call is very long – 17.553 ms. Average times vary with the type of network connectivity. For database calls with small packets sent and received, OSA Express Gigabit Ethernet is generally 1-3 ms, OSA-2 is generally 2-5 ms, and Escon generally 3-7 ms. 17ms is very long for any network type. Second, look at the distribution the packet sizes sent and received. If most of the packets are very large, then average times will be longer than these ROTs, because the large packets have to be broken down to network MTU size for sending. In this case, almost all packets are less than 512 bytes, so they will fit into the MTU.

Since average times are long, and small packet times are also long, there seems to be a problem in network performance. Follow-up actions would be checking CPU and paging activity on the DB server, checking WLM priority settings on the DB server, checking physical network settings and performance.

9.5. Check for global DB server problems

The DB2 administration guide (SC26-9003) contains detailed guidance for performance tuning with DB2. Following is a quick summary of key performance indicators on the database server.

9.5.1. CPU constraint

OS monitoring tools, such as RMF monitor I and monitor III, will report this problem. In addition, there are indicators in DB2, such as high "not attributed" time in ST04 "times", that can point to a CPU constraint on the database server.

RMF III, with the PROC command, can indicate the extent to which DB2 is delayed. The higher the PROC delay, the more DB2 can benefit from additional CPU resources.

Inefficient SQL can elevate CPU usage on the DB server, so the SQL cache should be examined as part of the action plan when a CPU constraint is seen on the DB server.

9.5.2. Bufferpool and hiperpool memory allocation

9.5.2.1. Hitrate goals

In SAP, each dialog step makes many database calls. In order to provide good dialog step response times, we want to achieve very high hitrates in the bufferpools. Since most R/3 transaction SQL is processed by DB2 as random getpages, the key indicator is "random hitrate" for most bufferpools. This is defined as *100* (random getpages – synchronous read random)/(random getpages)*, and is reported by ST04 and DB2PM. The random hitrate for most bufferpools should be in the high 90s.

For some bufferpools, such as BP1 (sorts) or bufferpools containing tables with primarily sequential access, random hitrate is not important. These bufferpools generally use prefetch I/O to access the data. Since random hitrate is not important here, but I/O throughput is, confirm that there are not I/O constraints on the disks, when examining the performance of bufferpools with primarily sequential access.

9.5.2.2. Bufferpool tuning

In order to match bufferpool attributes to the way tables are referenced, SAP and IBM provide guidelines for allocating tables to DB2 bufferpools. DB2 can allocate many bufferpools that are optimized to different access patterns. SAP manual 51014418 "SAP on DB2 UDB for OS/390 and z/OS: Database Administration Guide", describes how to analyze the table reference patterns, and place tables in bufferpools that have been created with attributes that match the access patterns. Use these guidelines to move tables with special or disruptive access patterns, for example large tables that have low re-reference rates like FI and CO tables used for reporting.

If you have good database performance (low DB2 delay percentage, etc) with the default bufferpool layout configured at installation, then there is probably no need to do the additional bufferpool tuning described in the SAP on DB2 UDB for OS/390 and z/OS: Database Administration Guide.

9.5.2.3. DB2 bufferpool memory with 31-bit real (up to OS/390 2.9)

Since storage in DBM1 is bounded by the 2GB address space VSTOR limit, and bufferpools are allocated from DBM1, hiperpools can be used to make more buffer memory available to DB2 for SAP. Hiperpools reside in page-addressable Expanded Storage (ES) outside the DBM1 address space. Since hiperpools cannot contain "dirty" pages (pages which have been changed, but not written to disk), bufferpools containing tables that are re-referenced and not frequently changed are good candidates for backing by hiperpools. Changed pages are written to disk before the page is written to hiperpool, and if pages in a table are seldom re-refererenced (as with large tables used for reporting) then prefetch I/O is the most effective way to bring the tables to DB2.

The key indicator for determining if hiperpools are being effectively used is the re-reference ratio. It is the percentage of pages written from bufferpool to the hiperpool that are later read back to the bufferpool. This is reported in SAP bufferpool detail statistics as "Hiperpool efficiency". A re-reference ratio above one in 10 (reported as 0.10 in hiperpool efficiency) means that the hiperpool is effective. If the re-reference ratio is lower, then the overhead of searching the hiperpool, failing to find the page, and having to do the I/O is not worth the savings gained on the few occasions when the page is found, and the hiperpool is not helping performance.

9.5.2.4. DB2 buffer memory with 64-bit real (z/OS and OS/390 2.10)

While DBM1 is currently still bounded by a 31-bit addressing limit, with 64-bit real support, dataspaces can be used (instead of a bufferpool and hiperpool pair) for each bufferpool. This has the advantage that DBM1 VSTOR constraint is alleviated, since the dataspace for the bufferpool resides outside DBM1. DBM1 contains control structures to reference the dataspace, which take much less memory than the size of the dataspace.

Unlike the bufferpool/hiperpool architecture, where hiperpools cannot contain dirty pages (changed pages not yet written back to disk), dataspaces offer a single pool that is managed in the same way as bufferpools. Dataspaces *must be backed by real storage (CS)*, not ES, for good performance.

9.5.3. DB2 sort

Due to the nature of SQL used with SAP R/3, which is generally simple indexed SQL, sort performance is very seldom a problem with R/3. With SAP R/3, monitor for the standard DB2 indicators – prefetch reduced or DM critical threshold reached, which show that the space in the bufferpool is not sufficient to satisfy demand. Check I/O performance on the volumes where the sortwk datasets are allocated, to verify that there is not an I/O constraint at the root cause.

DB2 sort can be a performance issue with BW systems, or APO systems, since the SQL for infocubes can be very complex. BP1 (the sort bufferpool for SAP) may need to be enlarged, and many SORTWK areas may be needed. SORTWK areas should be spread across several disks.

9.5.4. DB2 rid processing

RID processing is used by DB2 for prefetch processing (e.g. list prefetch, where non-contiguous pages are read in a single I/O) and for SQL processing (e.g. hybrid join).

There are four different kinds of problems related to RID processing, they are reported by SAP as:

- **RDS limit exceeded**, where the number of rids qualifying exceeds 25% of the table size. This is almost always the RID failure encountered with SAP, and is an indicator of a bad access path choice. The scenario is as follows. DB2 chooses an access path with RID processing at optimization. When executing the statement, DB2 recognizes that it is going to process more than 25% of the table, gives up on rid processing, and does a tablescan.
    - If this is causing an important performance problem, it can be addressed with FREQVAL RUNSTATS and ABAP hint, as described in Section 8.3. With RFCOSCOL-based ST04, RID failure statistics are available at the statement level. With earlier versions of SAP, one must use DB2 traces with IFCID 125 to find the SQL.
- **DM limit exceeded**, where the number of rids qualifying exceeds 2 million. This is really a variant of RDS limit exceeded, where RID processing of a huge table is being done. In this case, DB2 hits the DM limit before getting to RDS limit. The action is the same as RDS limit exceeded.
- **Storage shortage**, when the 2 GB VSTOR limit in DBM1 is hit. See SAPnote 162923 regarding VSTOR planning, and reduce the VSTOR demand by tuning MAXKEEPD, EDM, bufferpools, etc.
- **Process limit exceeded**, when the "RID pool size" configured in DB2 is exceeded. In this case, one can increase the size of the RID pool.

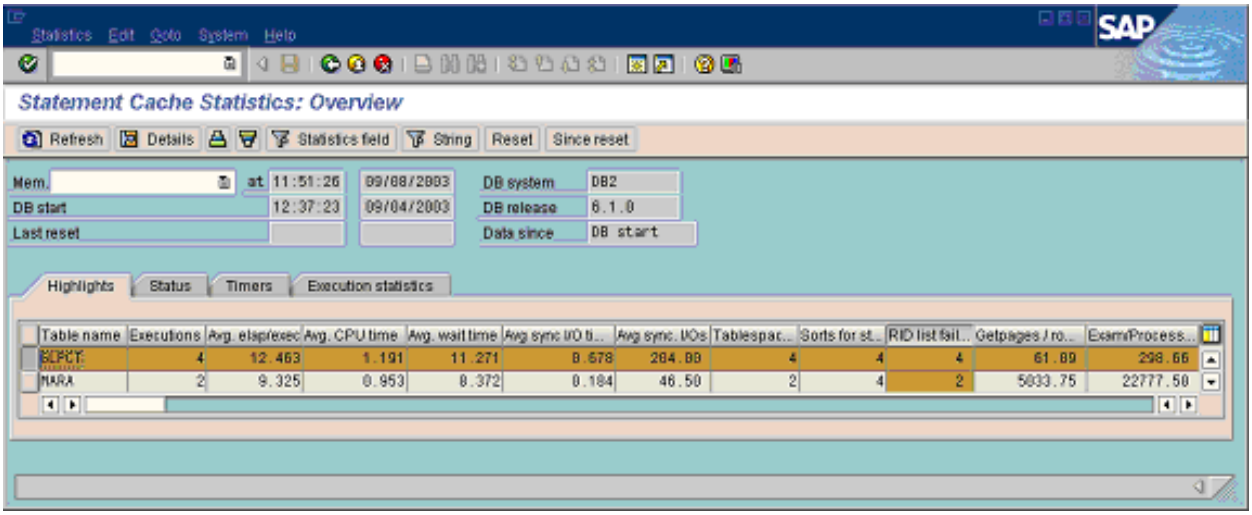


Figure 196: ST04 statement with RID failure

In Figure 196, the highlighted statement has had four RID list failures. The statement details will show the cause of the RID failure.

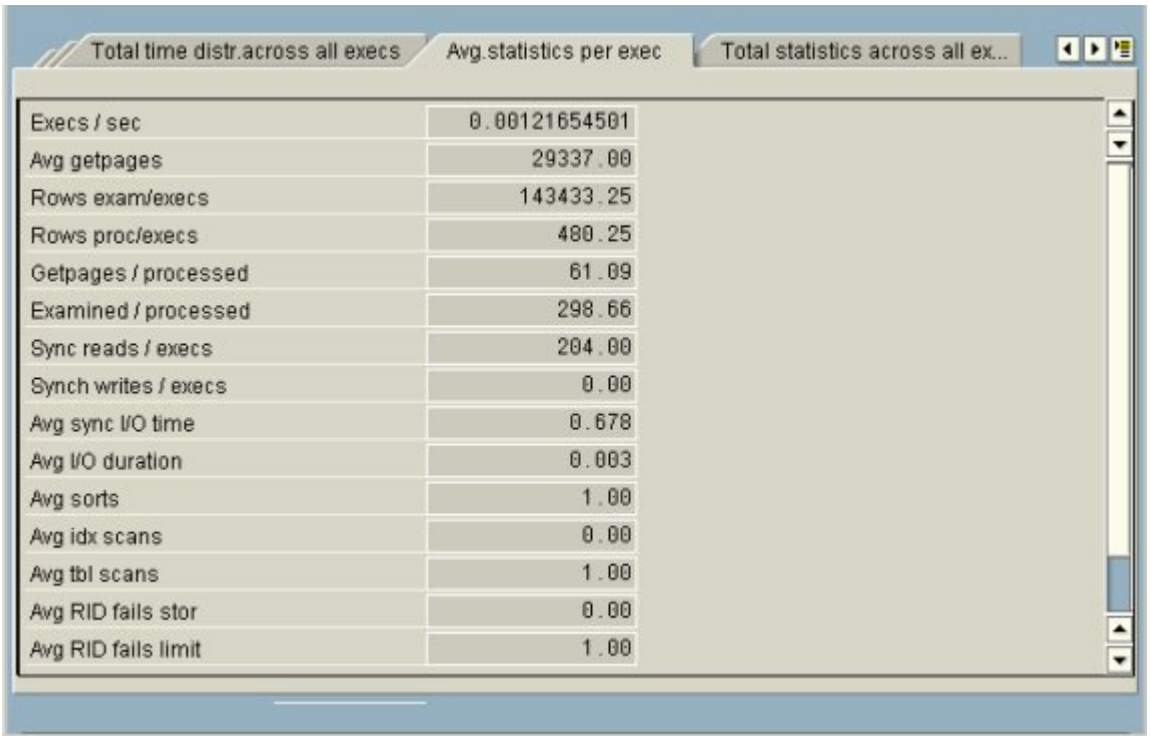


Figure 197: ST04 RID failure - statement statistics

Figure 197 shows that the RID failure was caused by “RID fails limit”, that is RDS limit exceeded, and that every statement executes a tablespace scan. When the statement is explained in Figure 198, we will see the access path that DB2 originally chose.

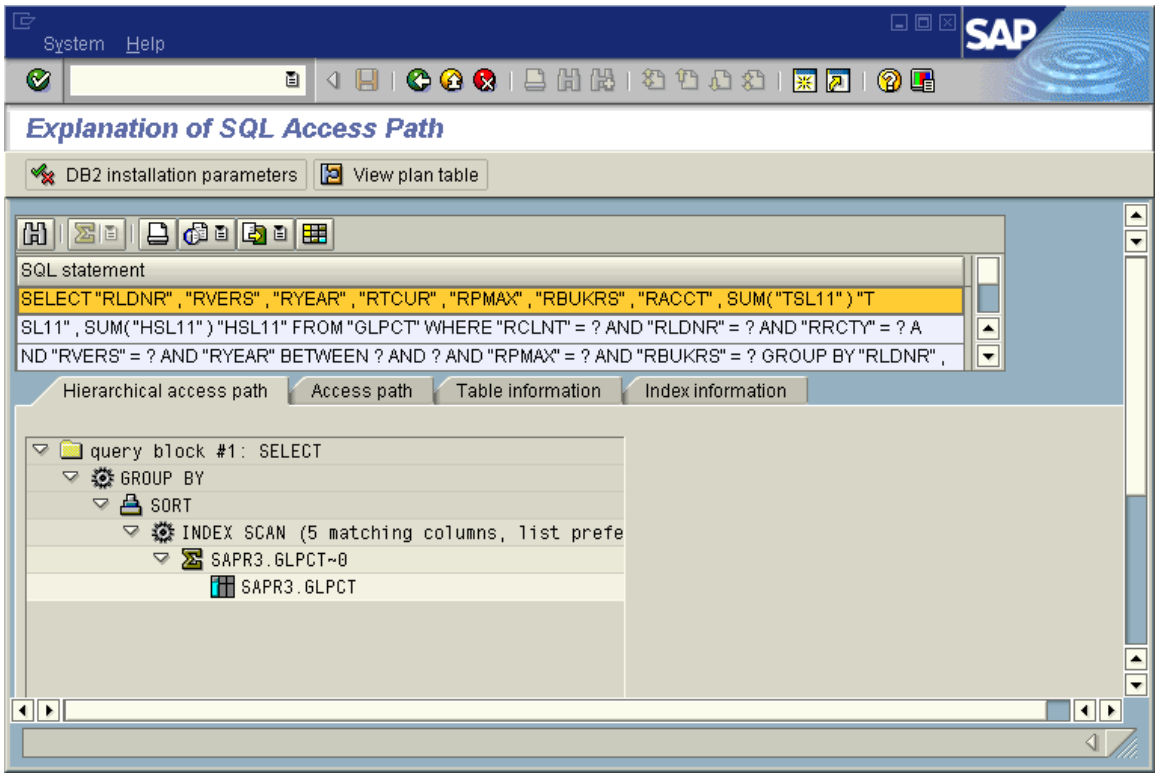


Figure 198: ST04 RID failure - DB2 access path before RDS limit failure

So, when reviewing the access path in ST04 explain, check the RID failures, to confirm whether DB2 actually used a different access path.

If this is an important performance problem, add an ABAP hint, and collect FREQVAL runstats, as described in Section 8.3

9.5.5. DB2 EDM and local statement cache

SAP uses DB2 dynamic SQL to access data. With dynamic SQL, the SQL is not bound in a plan, to be executed at runtime. The SQL is prepared at runtime. When statements are prepared using DB2 dynamic SQL, a skeleton copy of the prepared statement is placed in the EDM pool. Preparing the statement, and putting a skeleton copy in EDM is called a full prepare. The thread also gets an executable copy of the statement in its local statement cache.

Once the statement had been prepared and placed in the EDM pool, if another request to prepare the statement is issued by another DB2 thread (for an SAP work process), then DB2 can re-use the skeleton copy from EDM pool, and place an executable copy of the statement in the other thread's local statement cache. This is called a short prepare. Re-using a skeleton from the EDM pool takes about 1% as much CPU as the original prepare.

Each thread also has its own local copy of the statements that it is executing. The number of locally cached statements is controlled by the DB2 parameter MAXKEEPD.

The key indicators related to EDM and the statement cache are shown in ST04 “DB2 subsystem activity” as

- **Global hit ratio**, which is the hit ratio for finding statements from EDM pool when a statement is prepared. Since the original prepare is expensive, this should be kept high – 97%-99%, if possible.
- **Local hit ratio**, which is the hit ratio for finding statements in the local thread cache when a statement is executed. When the MAXKEEPD limit is hit, DB2 will take away unused statements from a thread’s local cache. If the thread then goes to use the statement which has just been stolen, DB2 will “implicitly prepare” the statement. If the global hit ratio is high, this implicit prepare will be quickly and efficiently done.

Focus on keeping the “global hit ratio” very high. If the “local hit ratio” is lower, even down to 60%-70%, it is not usually a problem for performance, since short prepares are very efficient.

If the “local hit ratio” is low, and there is VSTOR available in DBM1, then one can increase MAXKEEPD to increase the limit on the number of statements in the local cache. Since short prepares are very efficient, a low “local hit ratio” is not generally a problem. In general, it is better to keep MAXKEEPD at the default or below, and give VSTOR to DB2 buffers.

If VSTOR in DBM1 is constrained, and you are running DB2 on a system with 64-bit real hardware and software with sufficient real storage (CS), then the EDM pool can be moved to a dataspace. This will reduce the demand for VSTOR in DBM1.

Low local cache hit ratio is not generally not a problem in tuning an SAP DB2 system. Usually, the system installation defaults (or something a bit smaller) are fine. Tests done several years ago by IBM showed that 1,200 short prepares per minute increased CPU utilization by about 1%, compared to CPU utilization with 25 short prepares per minute. 20,000 short prepares per minute increased CPU usage by about 5%. These are samples based on older versions of SAP, and thus do not reflect real-world results, but they show that a system can run rather high rates of short prepares without a serious performance problem.

9.5.6. Memory constraint on DB server

The cardinal rule in allocating memory on the DB server is to adjust DB2 memory usage to avoid operating system paging on the DB server. It is more efficient to let DB2 do page movement between BP and HP than to have z/OS page between CS (central store) and ES (expanded store), or disk (aka aux storage).

Use the customary z/OS indicators in RMF, such as migration age (under 500-700) showing ES is overcommitted, UIC (under 60) showing CS is overcommitted, and migration rate (over 100) showing too much paging to disk.

9.5.6.1. ES constraint

Even without access to the z/OS monitoring tools, one can see symptoms of ES over commitment in DB2 indicators. See the “hpool read failed” and “hpool write failed” counters on the hiperpools. If these are more than a few percent of hiperpool page reads or writes, then there is probably an ES constraint that is causing z/OS to take ES pages from hiperpools. You can also see the impact of z/OS taking hiperpool pages away from DB2 in the bufferpool counters “hiperpool buffers backed” and “hiperpool buffers allocated”. If backed is less than allocated, it can also point to an ES constraint.

9.5.6.2. CS constraint

Without access to z/OS tools, one can see symptoms of CS constraint in DB2 indicators. High ST04 “not attributed” time, which is discussed in section 8.1, is an indicator of a possible CS constraint. In addition, the bufferpool counters “page-ins required for read” and “page-ins required for write” will indicate CS constraint. These should be very small, as a percentage of getpages – one percent at most. The usual goal with SAP is to have bufferpool hitrates in the high 90s. If there are also a few percent of getpages that have to be paged in, this in effect reduces the hitrate, and may decrease the bufferpool hitrate below the recommended range.

9.6. Sample global DB server problems

9.6.1. Example of ES constraint on DB server

Here is an RMF I report. ES storage constraint is indicated by a low migration age (MIGR AGE). The average is at the edge of our 500-700 ROT. MIGR AGE below this threshold shows over-commitment of ES.

```
                                    P A G I N G   A C T I V I T Y
                                                                                                                    PAGE    2
           OS/390                    SYSTEM ID TCP1            START 07/13/2001-15.00.00  INTERVAL 000.44.59
           REL. 02.09.00             RPT VERSION 2.7.0         END   07/13/2001-15.44.59  CYCLE 1.000 SECONDS
OPT = IEAOPT00                           EXPANDED STORAGE MOVEMENT RATES - IN PAGES PER SECOND
-------------------------------------------------------------------------------------------------------------------------------
     ESF CONFIGURATION                                                                                          HIGH UIC  MIGR AGE
   INSTALLED      ONLINE                                                                                  MIN   254       12
   ---------      ------                                                                                  MAX   254     1299
    258048       258048                                                                                   AVG   254.0    688.3
                                                  MIGRATED       FREED
                        WRITTEN TO     READ FROM      FROM     WITHOUT      *----- EXPANDED STORAGE FRAME COUNTS -----*
                          EXP STOR      EXP STOR    EXP STOR  MIGRATION            MIN          MAX          AVG
  TOTAL          RT         225.78         78.68       43.91      54.72        255,380      257,898      257,579
   PAGES          %         100.0%        100.0%      100.0%
  HIPERSPACE     RT          44.23          0.00        0.00                   103,569      124,663      113,695
   PAGES          %         19.6%          0.0%        0.0%
  VIO            RT           0.00          0.00        0.00                         0            1            0
   PAGES          %          0.0%          0.0%        0.0%
  SHARED         RT           0.00          0.00
   PAGES          %          0.0%          0.0%
```

Figure 199: RMF I Paging report – ES constraint

9.6.2. Example of CS constraint on DB server

In this example from an RMF I report, there is too little CS available on the database server, as shown by the UIC being somewhat low (AVG 33, MIN 5, rule-of-thumb 60), while there is no constraint on ES, since the ES migration age is high. Since is it more efficient for DB2 do page movement between CS and ES by moving between bufferpool and hiperpool, it would be preferable to reduce the overall demand on CS and move storage demand to ES. One could do this by reducing the size of the size of the DB2 bufferpools, and increasing the size of the hiperpools, or by reducing MAXKEEPD.

```
                                P A G I N G   A C T I V I T Y
                                                                                                                    PAGE    2
           OS/390                    SYSTEM ID VSP             START 03/27/1999-14.00.00  INTERVAL 001.00.00
           REL. 02.05.00             RPT VERSION 2.4.0         END   03/27/1999-15.00.00  CYCLE 1.000 SECONDS
OPT = IEAOPT00                           EXPANDED STORAGE MOVEMENT RATES - IN PAGES PER SECOND
-------------------------------------------------------------------------------------------------------------------------------
     ESF CONFIGURATION                                                                                          HIGH UIC  MIGR AGE
   INSTALLED      ONLINE                                                                                  MIN      5    99373
   ---------      ------                                                                                  MAX     70   104646
    925696       925696                                                                                   AVG    33.1 102007.9
                                                    MIGRATED       FREED
                        WRITTEN TO     READ FROM        FROM     WITHOUT      *----- EXPANDED STORAGE FRAME COUNTS -----*
                          EXP STOR     EXP STOR    EXP STOR  MIGRATION              MIN          MAX          AVG
  TOTAL          RT        421.52       399.90        0.00        0.00          359,382      374,664      369,330
   PAGES          %        100.0%       100.0%      100.0%
  HIPERSPACE     RT          0.00         0.01        0.00                      132,000      132,000      132,000
   PAGES          %          0.0%         0.0%        0.0%
  VIO            RT          0.00         0.00        0.00                            8           10            9
   PAGES          %          0.0%         0.0%        0.0%
  SHARED         RT          0.00         0.06
   PAGES          %          0.0%         0.0%
```

Figure 200: RMF I Paging report – CS constraint

9.6.3. CPU constraint

One can use OS07 to view a snapshot of current CPU use in the LPAR on the DB server. OS07 shows that at this moment the LPAR is using 100% of its available CPU on the system (z/OS CPU).

The system CPU shows the utilization of the logical processors (LPs) defined to the LPAR, but if an LPAR has used all the CPU it is entitled to by its LPAR weighting,, then system CPU can show a lower utilization, while z/OS CPU shows 100% usage.

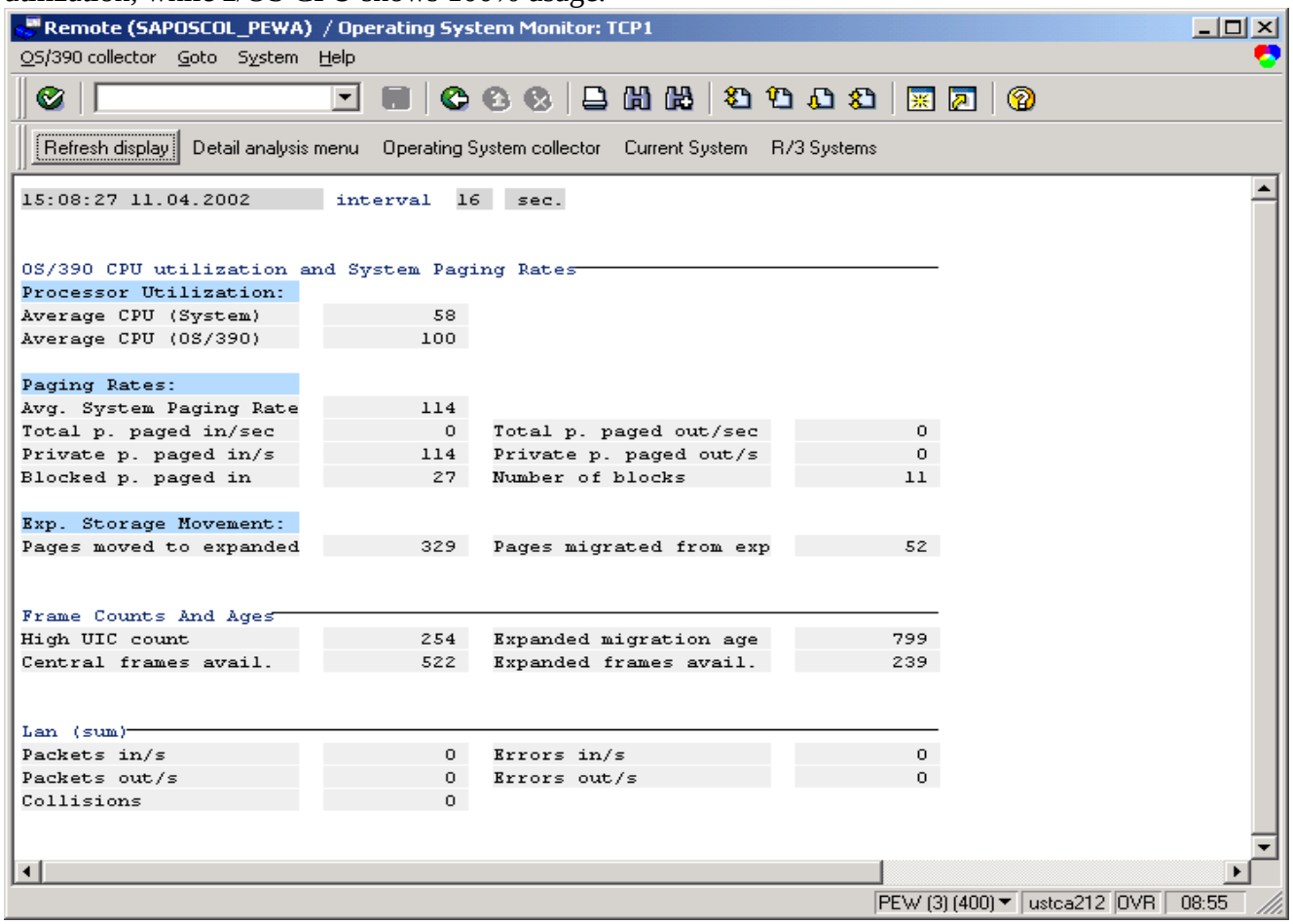


Figure 201: OS07 - overview of DB server performance metrics

One can use tools such as RMF I and RMF III to view recent periods, or longer periods.

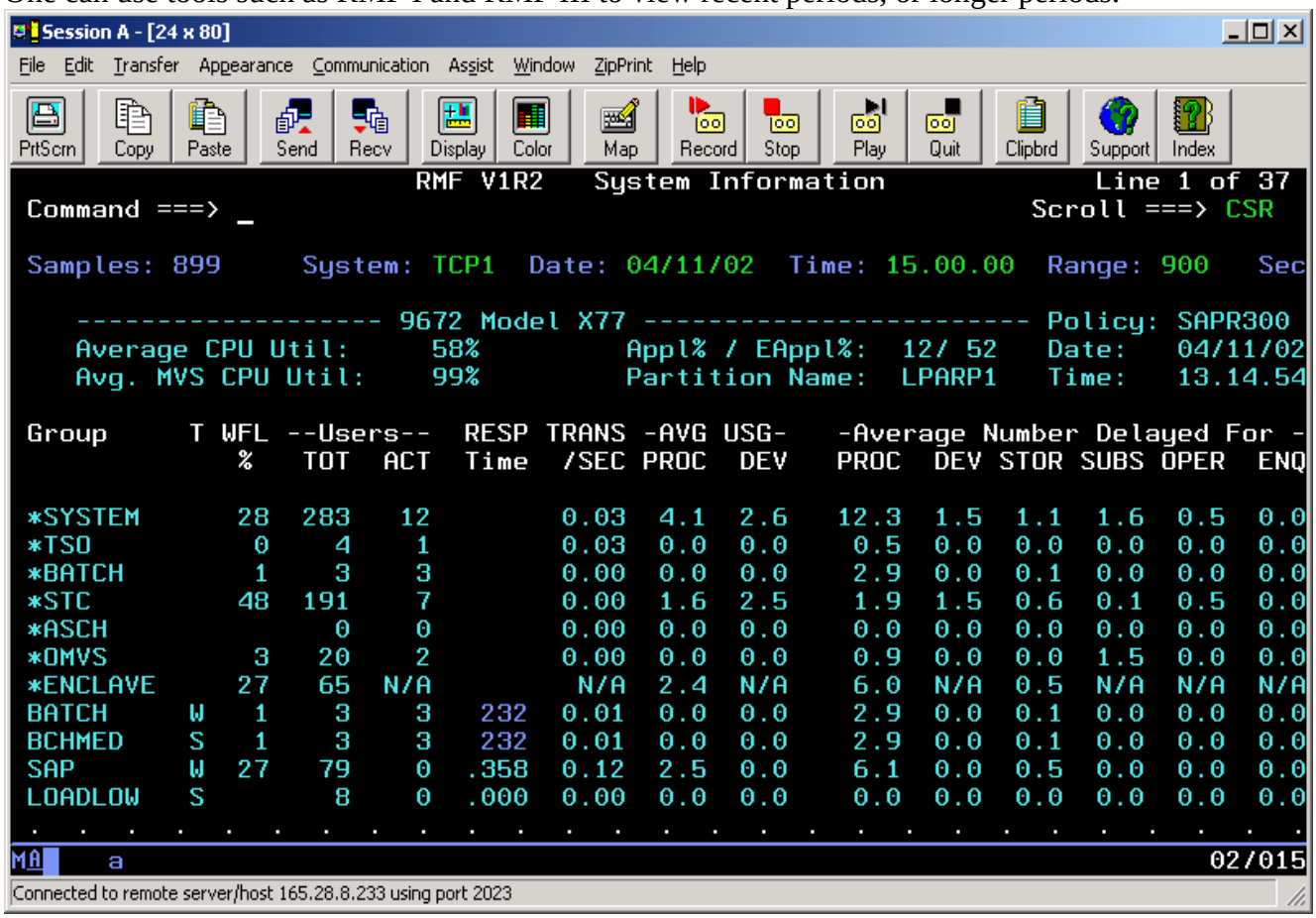


Figure 202: RMF III SYSINFO of 900 second interval

Log into DB server, and use RMF III SYSINFO command, which here shows 99% CPU utilization over a 900 second ("range") period. "Avg. MVS CPU Util." shows the utilization of CPU available to this LPAR.

In this case, the CEC (physical zSeries box) was at 100% CPU utilization, so the LPAR was constrained by its CPU weighting relative to other LPARs, though the LPs defined to the LPAR were only active 58% of the time.

9.6.4. I/O constraint

In this example, there is a system constraint caused by a configuration problem. As on some of the other examples, this is not a problem one would expect to find in real life, but the process shows how to go from SAP performance indicators to z/OS statistics.

Start with an ST03 workload summary. Generally, ST03 is not helpful in showing performance problems, but if the average times for sequential read, changes, or commit are exceptionally high, it can

point to a problem on the DB server. In this case, average commit time is over 50 ms, which is unusual. It could be normal, where there are jobs that are making many changes before commit, or it could be a sign of a problem.

-Workload of tasktype *TOTAL*

CPU time	1,853.0	s	Database calls	955,076	
Elapsed time	3,600.0	s	Database requests	4,173,151	
			Direct reads	3,215,016	
Dialog steps	75,593		Sequential reads	667,195	
			Changes	290,940	
Av. CPU time	24.5	ms			
Av. RFC+CPIC time	0.1	ms	Time per DB request	5.4	ms
			Direct reads	0.9	ms
Av. response time	575.3	ms	Sequential reads	4.8	ms
Av. wait time	27.9	ms	Changes & commits	56.2	ms
Av. load+gen time	2.7	ms			
Av. roll time	0.1	ms	Roll in time	5.8	s
Av. DB req. time	296.4	ms	Roll out time	0.9	s
Av. enqueue time	0.0	ms	Roll wait time	2,019.4	s
			Roll ins	9,249	
Av. bytes req.	0.2	kB	Roll outs	326	

Figure 203: ST03 with long change and commit time

The ST04 times display is from a DB2 V5 system, where “service task switch” contains commit processing. Here, class 2 is not gathered, but class 3 is. Note that the average “service task switch” is 93 ms, which is long. In DB2 V5, commit processing s part of “ServTaskSwitch” suspension.

I/O suspension time	0.000	I/O susp (Avg)	6.72
Lock/Latch susp time	0.000	Lock/Latch susp (Avg)	6.513
Other Read Susp	0.000	Othr read susp (avg)	7.961
Othr write susp.	0.000	Othr write susp (avg)	65.839
ServTaskSwitchsusptime	0.000	ServTaskSwitchSusp (Avg)	93.55
Arch log quiesce susp	0.000	ArchLogQuiesceSusp (Avg)	0
Drain lock susp time	0.000	Drain lock susp (avg)	0
Claim rel susp time	0.000	Claim rel susp (Avg)	0
Arch read susp time	0.000	Arch read susp (Avg)	0
Page latch susp time	0.000	Page latch susp (avg)	45.29
Notify mess susp time	0.000	Notify mess susp (Avg)	0
Global lock susp time	0.000	Global lock susp (avg)	0

Figure 204: ST04 times long service task switch

Now, go to z/OS RMF III. Check DEV to see device delays.

```
                    RMF 2.4.0  Device Delays                    Line 1 of 4

 Samples: 3589    System: SAPS  Date: 07/23/99  Time: 17.00.00  Range: 3600  Sec

            Service  DLY USG CON  ------------ Main Delay Volume(s) ------------
 Jobname  C Class     %   %   %    %  VOLSER   %  VOLSER   %  VOLSER   %  VOLSER

 QASAMSTR S DB2D      64  77 155   61 QAS001    4 QAS002    1 TMM019    1 TMM023
 QASADBM1 S DB2D       8  32  70    1 QAS047    1 QAS027    1 QAS038    1 QAS013
 CATALOG  S SYSTEM     2   1   1    2 QAS001    0 MCT501    0 QAS053    0 QAS013
 OAM1     S SYSSTC     1   1   0    1
```

Figure 205: RMF III DEV report

There was one device, QAS001, causing the most I/O delay to DB2. The address space, QASAMSTR, is a hint that this is not a problem with I/O to the tables and indexes in the DB2 database. RMF III credits delay on tables and indexes to DBM1.

Next, look at the datasets on the QAS001 volume.

```
                    RMF 2.4.0  Data Set Delays - Volume          Line 1 of 8

 Samples: 3589    System: SAPS  Date: 07/23/99  Time: 17.00.00  Range: 3600  Sec

 -------------------------- Volume QAS001 Device Data --------------------------
 Number:     1370         Active:      89%        Pending:     2%     Average Users
 Device:     33903        Connect:     72%        Delay DB:    0%        Delayed
 Shared:     Yes          Disconnect:  15%        Delay CU:    0%          0.6
                                                 Delay DP:    0%
 -------------- Data Set Name ---------------    Jobname   ASID  DUSG% DDLY%
 SAPQASA.LOGCOPY1.DS02.DATA                      QASAMSTR  0087     22    26
 SAPQASA.LOGCOPY1.DS01.DATA                      QASAMSTR  0087     23    23
 SAPQASA.LOGCOPY1.DS03.DATA                      QASAMSTR  0087     19    22
 SYS1.VVDS.VQAS001                               CATALOG   0045      0     1
 SAPQASA.BSDS02.INDEX                            QASAMSTR  0087      0     1
 SAPQASA.BSDS02.DATA                             QASAMSTR  0087      0     1
 CATALOG.SAPQAS                                  CATALOG   0045      0     0
 CATALOG.SAPQAS.CATINDEX                         CATALOG   0045      0     0
```

Figure 206: RMF III DSNV report

Note that the volume contains three log datasets. When a log switch occurs, DB2 is writing the log to one dataset, and copying the log from another on the same disk.

This was a QA system, which had not been setup using the standard guidelines for productive systems. On a productive system, logs should be on separate volumes. One would not expect to see this problem on a productive system. The goal of the exercise was to link the SAP ST03 change and commit time down to the z/OS cause.

10. Estimating the impact of fixing problems

10.1. ST04 cache analysis

Start cache analysis by using ST04 sorted by total getpages, rows examined, or elapsed time. In this way, you can focus on statements with the largest impact on the system.

10.1.1. Estimating the opportunity for improvement in inefficient SQL

When evaluating inefficient statements in the SQL cache, one can estimate the potential improvement in resource usage and response time, by comparing the current resource usage with hypothetical good SQL statement usage.

For example, you have found a statement that does 500 getpages per row returned and takes 100 ms internal DB2 time (ST04 average elapsed time). One can estimate that if the could be converted into a well indexed statement where all rows could be selected based on an index, that it would take just a few getpages per execution, and have an internal DB2 time of under 1 ms. We have seen many examples of how efficiently indexed SQL takes only a few getpages for each row returned. If an inefficient statement runs thousands of times a day (see ST04 statement statistics for counters), then fixing the problem would help overall system performance. Addressing problems of this sort will help to improve bufferpool hitrates, reduce I/O, and improve system response time.

The kinds of problems usually seen, and the order in which we suggest to address them is:

- Complaints of end-users.
- Frequently executed inefficient statements. Inefficient code in frequently run programs such as transaction user exits can both slow transaction performance and impact system performance. These statements might take tens to hundreds of milliseconds to run, and perform hundreds to thousands of getpages per row processed. Either ABAP changes (hints or code changes) or new indexes are justifiable in these cases.
- Periodically run very inefficient statements. For example, an interface or reporting program that runs many times throughout the day. In cases such as these, there could be very inefficient sql (hundreds or thousands of getpages per row). Fixing these can also reduce resource utilization. These might be fixed via ABAP, but if an index can be created that is small, filters well, and is in a reasonable location (e.g. on a header table rather than document table) then an index could be justified, too. Here the benefit to the system is less than the benefit in fixing frequently executed statements, so one should be more cautious about adding an index, in order to avoid index proliferation and space usage.
- Really bad SQL (tens of thousands or hundreds of thousands of getpages per SQL) in jobs that run just a few times a day or week. If these can be fixed with SQL changes, then the programmers can prioritize the work, based on the impact of the fix, and the business need for better performance. If the problem cannot be fixed by ABAP, and requires a new index, it is usually not worthwhile to create a new index (given the tradeoff between disk space used and performance benefit gained),

unless there is a critical business need for better performance. In this case, though it is inefficient, let DB2 and the operating system manage it.

10.2. ST10 table buffering

The most common problem is bufferable tables that are not buffered. As described above, bufferable tables are tables that are

- Mostly read-only
- Moderate size
- The application can tolerate a small interval where the buffered data is different than the database

Compare the calls and rows fetched by the candidate table to the total calls and rows for the reporting interval. If the table makes up more than 0.5% to 1% of total call or row activity, then we would suggest buffering it.

The benefit of buffering a table is related to, but will not be the same as its percentage of calls or rows. That is, buffering a table with 5% of calls will not offload 5% of the database server, since complex SQL and inefficient SQL uses much more CPU per call than simple indexed SQL.

The most notable impact of changing table buffering will be on the transactions and programs that read the table. Buffered table reads usually take less than 0.1 ms per row. If a transaction is reading many rows from the table, then the benefit for the transaction will be proportional to the number of rows read.

11. Four guidelines for avoiding performance problems

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

11.1. 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. 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 example in Section 8.5.3, 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 buffer gets per exec and high buffer gets per row. This happens when the predicates on the SQL do not match the index columns on the table and Oracle has to read many blocks of data to retrieve the result.

11.2. 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.

11.3. 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.
- 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.
- A program is repeatedly fetching the same information from the database. This problem can be detected by using the 'duplicate selects' function 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

11.4. 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
- Evaluate whether internal tables need to be defined as SORTED

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.

12. How to tell when you are making progress

12.1. SAP

Normally, it is best to view improvement from the application, by using STAT or ST03(N) to evaluate the elapsed time. SM37 or SM39 can be used to track elapsed time for batch jobs by the batch job name, rather than the name of the program executed.

Since the gains for an individual program can be very dramatic with tuning, the reduction gained in elapsed time for programs will generally be much more notable than overall improvements in section 12.2

12.2. DB2 and S/390

Using DB2 or z/OS indicators to measure progress is more challenging, due to the variable nature of the SAP workload, and the way that transaction and batch workload run together. A batch job is counted as one dialog step, and may do thousands or millions of SQL operations. When this SQL activity is averaged into DB server statistics, a few batch jobs can have a dramatic impact on CPU utilization, without making a significant change to dialog step counts. Thus, our preference for the application view – STAT records, ST03, etc. If you are working on improving the efficiency of SQL on the system, there will generally also be reductions in

- CPU utilization
- I/O activity rates

Since the dialog steps per hour can vary widely from day to day, and improving SQL can change the amount of CPU used by a statement, one can look at other ways to normalize work in terms of DB2 work performed, such as reductions in

- Getpages per SQL DML (calculated from DB2PM) – with SQL improvements, DB2 searches fewer pages to return the result
- CPU per dialog step – this is a “dialog step normalized” view of reduced CPU utilization
- CPU per SQL DML operation – an “SQL normalized” view of reduced CPU utilization

It’s generally simplest to stick to reduction in transaction elapsed time. If there has been a large effort to improve SQL efficiency, then there should be reduced CPU utilization for the same number of dialog steps, if the workload mix does not change.

13. Appendix 1: summary of performance monitoring tools

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

13.1. SAP

13.1.1. 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 > detail analysis)
- Check index and table cardinality (DB02 > detail analysis > enter table name > drill into table > drill into index)

13.1.2. DBACOCKPIT Transaction

DBACOCKPIT integrates the database storage and performance management transactions. All the ST04 functions described in this paper are also available from DBACOCKPIT.

13.1.3. 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 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
- Sites using DB2 V5 or V6 use “where used” to find programs that reference a table or view. There are some gotchas with “where used”:
    - SAP Dynamic SQL, where the statement is constructed at runtime by the ABAP, may not be found in where used
    - The SQL in ST04 cache may not match SQL in program. E.g. when the user can optionally enter parameters for several predicates, only the predicates that are specified will be in the executed SQL.
    - Sites using DB2 V7 and SAP 6.20 do not need to use “where used”, since statements in ST04 statement cache have a marker with the ABAP program name.

13.1.4. SE30 Transaction

When STAT or ST03 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

13.1.5. 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.

13.1.6. 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.

13.1.7. 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.
- If there are queues for ENQ, there is a problem with enqueue performance.

13.1.8. SM66 Transaction

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.

13.1.9. STAD Transaction

Is used to displays STAT records for an interval from all instances on an SAP system. It aggregates the RFC call information, so is not as useful as STAT in finding problems with slow RFCs.

13.1.10. ST02 Transaction

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

13.1.11. ST03 Transaction

ST03 is not a tool for solving performance problems. Like RMF I, it is a tool which 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 ST03 “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 ST03 “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.

- Use as a filter for database performance problems, in very limited circumstances. If average "sequential read" times are over 10 ms for dialog, and commit time is over 25-30 ms, there may be some sort of database performance problem. Check SQL cache with ST04, look for I/O constraints and other database problems.

13.1.12. ST04 Transaction

ST04 has many functions, the most important for performance are viewing the SQL cache, checking DB2 delays, and monitoring bufferpool activity and monitoring DB2 threads.

13.1.13. 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.

13.1.14. ST06 Transaction

Display OS level stats for the application server – paging, CPU usage, and disk activity. The statistics displayed are incorrect if the application is AIX SPLPAR.

13.1.15. ST06N

If using AIX application servers with SPLPAR, ST06N with the saposcol version in SAP note 994025 are required to monitor CPU use correctly.

13.1.16. 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

13.1.17. ST12

ST12 enables simultaneous ST05 and SE30 executions, so that both DB and application server activity for a transaction can be traced together.

13.1.18. 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.

13.1.19. 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, or program SQLR0001.

13.1.20. 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 or Thread ID to establish the trace. Once the user is locked into a work process, one can determine the PID and Thread ID, and use them to establish the trace.

13.2. z/OS

13.2.1. RMF I

Is a tool for historical reporting, and is useful for tracking capacity planning related information, such as CPU activity, and I/O activity. It can also be useful for trending and monitoring OS level constraints such as CPU or memory. Since the disk information shows volume level activity, and there can be many DB2 datasets on a volume, the DASD information needs to be augmented by RMF III (or some other real-time analysis tool) to find the datasets causing delays, so that the SQL can be found and analyzed.

13.2.2. RMF II

The RMF II SPAG command has a good summary of information related to paging, but it must be gathered real-time.

13.2.3. RMF III

RMF III is a powerful tool that can be used for real-time analysis, as well as for reporting recent history. Use it to determine the causes of DB2 delays (e.g., which volume is causing I/O delay) when drilling down from SAP. Just a few functions are sufficient to diagnose the usual problems seen with SAP and DB2:

- SYSINFO – CPU utilization information
- DELAY – summary of causes of delays
- DSNJ (for the DBM1 address space) – show datasets causing delays
- DEV – show devices causing delays
- DEVJ (for the DBM1 address space) show volumes causing delays
- DEVR – show I/O rates and average response times
- PROC – show processor delays
- STOR – show storage delays

13.3. DB2

The most important DB2 performance tool, statement cache analysis, is in SAP ST04 transaction.

13.3.1. DB2PM

- STATISTICS is focused on activity, and not delays. It often shows symptoms (e.g. low hit rate, thresholds being exceeded), not causes (inefficient SQL or programs not committing). After SQL has been addressed, then monitor the DB2 indicators of bufferpool hit rates, EDM pool activity, etc. Note that the random hit rate is usually the key metric for DB2 buffer pool hit rate. Older versions of DB2PM, such as V5, do not calculate random hit rate.

- ACCOUNTING is focused on time in DB2 and delay in DB2, but since there is no easy way to correlate a DB2 thread to an SAP job, and since many different SAP transactions will execute in the same thread, and threads are restarted periodically, it has limited use. One can aggregate the thread statistics to view the overall sources of DB2 delays, as the ST04 “times” transaction does.
- For systems which do not have RFCOSCOL ST04, IFCID traces may be needed to find lock, latch,and RID failure problems:
  - Use IFCID 44,45,226, and 227 to investigate problems with lock and latch suspensions.
  - Use IFCID 125 to investigate problems with RID failures.

14. Appendix 2: Reference Materials

14.1. SAP Manuals

51014418 “SAP on DB2 UDB for OS/390 and z/OS: Database Administration Guide”

SAP Planning Guide for SAP NetWeaver on IBM DB2 for z/OS (NW04)

SAP Planning Guide for SAP NetWeaver on IBM DB2 for z/OS (NW04S)

14.2. IBM manuals

Planning Guides, which contain detailed description of architecture of SAP to DB2 connection:

SC33-7961-02 “SAP R/3 on DB2 for OS/390: Planning Guide SAP R/3 Release 3.1I”

SC33-7962-02 “SAP R/3 on DB2 for OS/390: Planning Guide SAP R/3 Release 4.0B”

SC33-7962-03 “SAP R/3 on DB2 for OS/390: Planning Guide SAP R/3 Release 4.0B SR 1”

SC33-7964-00 “SAP R/3 on DB2 for OS/390: Planning Guide SAP R/3 Release 4.5A”

SC33-7964-01 “SAP R/3 on DB2 for OS/390: Planning Guide SAP R/3 Release 4.5B”

SC33-7966-00 “SAP R/3 on DB2 for OS/390: Planning Guide SAP R/3 Release 4.6A”

SC33-7966-01 “SAP R/3 on DB2 for OS/390: Planning Guide SAP R/3 Release 4.6B”

SC33-7966-02 “SAP R/3 on DB2 for OS/390: Planning Guide SAP R/3 Release 4.6C”

SC33-7966-03 “SAP R/3 on DB2 for OS/390: Planning Guide SAP R/3 Release 4.6D”

SC33-7959-00 “SAP on DB2 UDB for OS/390 and z/OS: Planning Guide” (SAP Web AS 6.10)

SC33-7959-01 “SAP on DB2 UDB for OS/390 and z/OS: Planning Guide” (SAP Web AS 6.20)

DB2 Administration Guides, which contain detailed description of DB2 access paths, prefetch capabilities, buffer pool parameters, and components of DB2 elapsed time:

SC26-8957-03 “DB2 for OS/390 Version 5: Administration Guide”

SC26-9003-02 “DB2 Universal Database for OS/390: Administration Guide” (DB2 V6)

SC26-9931-01 “DB2 Universal Database for OS/390 and z/OS: Administration Guide” (DB2 V7)

SC18-7413-05 “DB2 UDB for z/OS V8 Administration Guide”

SC18-9840 “DB2 Version 9.1 for z/OS Administration Guide”