



***IBM SAP Technical Brief***

**Tuning SAP / DB2 / zSeries**

**Mark Gordon**

**IBM Solutions Advanced Technical Support**

**Version: 1.0**  
**Date: June 1, 2002**

---

1.	Acknowledgements.....	5
2.	Disclaimers .....	5
3.	Copyrights.....	5
4.	Feedback .....	5
5.	Introduction.....	5
6.	DB2/390 and SAP background information .....	7
6.1.	SAP .....	7
6.2.	DB2.....	7
7.	Solving a performance problem for a specific program or transaction .....	9
7.1.	Check STAT records for components of SAP elapsed time.....	9
7.1.1.	Description of STAT record time components.....	9
7.1.1.1.	Description of STAT “missing time” .....	10
7.1.1.2.	Description of STAT detailed database request time .....	11
7.1.1.3.	Description of <i>stat/tabrec</i> and <i>rsdb/stattime</i> parameters .....	13
7.2.	Actions from STAT record analysis .....	15
7.3.	Examples of problem indicators in STAT records .....	17
7.3.1.	Low CPU time example.....	18
7.3.2.	High CPU time example .....	18
7.3.3.	High RFC+CPIC time example .....	19
7.3.4.	Response time .....	20
7.3.5.	Wait for work process example .....	20
7.3.6.	Processing time examples.....	22
7.3.7.	High load time example.....	24
7.3.8.	Roll (in+wait) time example .....	25
7.3.9.	Database Request time.....	26
7.3.10.	Enqueue examples .....	26
7.3.11.	Frontend example.....	32
7.3.12.	Missing time in STAT – suggested actions .....	33
7.4.	Transaction.....	34
7.4.1.	Analysis process for transactions.....	34
7.4.2.	Sample transaction analysis - MIRO .....	34
7.4.3.	Sample transaction analysis – ME21L.....	48
7.4.4.	Sample transaction analysis - MB51.....	58
7.5.	Batch .....	65
7.5.1.	Analysis process for batch .....	65
7.5.2.	Batch analysis starting from STAT records.....	66
7.5.3.	Sample of SQL cycle analysis in batch.....	69
7.5.4.	Sample end-to-end batch time analysis.....	74
8.	Check for inefficient use of DB resources.....	77
8.1.	DB2 accounting data – delay analysis .....	77
8.1.1.	Components of DB2 delay .....	79
8.1.2.	Key indicators in DB2 Times .....	80
8.1.3.	Actions to take for DB2 times indicators.....	80
8.2.	DB2 delay analysis examples .....	82
8.2.1.	Good ST04 times .....	82

8.2.2.	Rather suspicious ST04 times.....	83
8.2.3.	ST04 Times points to constraint on DB server.....	84
8.3.	Process for Identifying slow or inefficient SQL.....	86
8.3.1.	High getpages and low rows processed (per execution).....	89
8.3.2.	High rows examined and low rows processed (per execution).....	90
8.3.3.	Long “average elapsed time” per execution.....	90
8.4.	Examples of searching for slow or inefficient SQL.....	90
8.4.1.	Using SE11 “where used”.....	90
8.4.2.	Predicates do not match available indexes.....	98
8.4.3.	Incorrect use of SAP tables.....	105
8.4.4.	Impact of dynamic SQL on access path chosen by DB2.....	113
8.4.5.	I/O constraint on table.....	125
8.4.6.	Index screening.....	128
8.4.7.	Growing pains - catalog statistics out of date.....	133
8.4.8.	Evaluating whether a new index is needed.....	142
8.4.9.	Logical row lock contention.....	148
8.4.10.	DB2 page latch contention.....	149
9.	Health Check.....	151
9.1.	Check for SAP instance-level or system-level problems.....	151
9.1.1.	Application server OS paging.....	151
9.1.2.	Application server CPU constraint.....	151
9.1.3.	SAP managed memory areas.....	151
9.1.4.	Table buffering.....	152
9.1.5.	Wait time.....	152
9.1.6.	Number ranges.....	153
9.2.	Sample SAP instance-level and system-problems.....	154
9.2.1.	Application server paging.....	154
9.2.2.	Application Server CPU constraint.....	155
9.2.3.	Roll Area shortage.....	157
9.2.4.	ST02 buffer area shortage.....	159
9.2.5.	Find table buffering candidates.....	160
9.2.6.	Table buffered with wrong attributes.....	162
9.2.7.	Number range buffered by quantity that is too small.....	164
9.3.	Check for network performance problems.....	167
9.3.1.	Lost packets indicators.....	167
9.3.2.	Slow network indicators.....	168
9.4.	Sample network performance problems.....	168
9.4.1.	Slow network performance example.....	168
9.4.2.	Lost packets example.....	170
9.5.	Check for global DB server problems.....	170
9.5.1.	CPU constraint.....	171
9.5.2.	Bufferpool and hiperpool memory allocation.....	171
9.5.3.	DB2 sort.....	172
9.5.4.	DB2 rid processing.....	172
9.5.5.	DB2 EDM and local statement cache.....	173
9.5.6.	Memory constraint on DB server.....	174

9.5.6.1.	ES constraint .....	174
9.5.6.2.	CS constraint .....	174
9.6.	Sample global DB server problems .....	175
9.6.1.	Example of ES constraint on DB server .....	175
9.6.2.	Example of CS constraint on DB server .....	176
9.6.3.	CPU constraint .....	176
9.6.4.	I/O constraint .....	176
	Estimating the impact of fixing problems .....	179
9.7.	ST04 cache analysis .....	179
9.7.1.	Estimate system impact of inefficient SQL .....	179
9.7.2.	Estimating the opportunity for improvement in inefficient SQL .....	183
9.8.	ST10 table buffering .....	184
9.9.	STAT- evaluating performance in an identified program .....	185
10.	How to tell when you are making progress .....	187
10.1.	SAP .....	187
10.2.	DB2 and S/390 .....	187
11.	Appendix 1: summary of performance monitoring tools .....	188
11.1.	SAP .....	188
11.1.1.	DB02 .....	188
11.1.2.	SE11 .....	188
11.1.3.	SE30 .....	188
11.1.4.	SM12 .....	188
11.1.5.	SM50 .....	189
11.1.6.	SM51 .....	189
11.1.7.	SM66 .....	189
11.1.8.	STAT .....	189
11.1.9.	STAD .....	189
11.1.10.	ST02 .....	189
11.1.11.	ST03 .....	189
11.1.12.	ST04 .....	190
11.1.13.	ST05 .....	190
11.1.14.	ST06 .....	190
11.1.15.	ST10 .....	190
11.1.16.	RSINCL00 .....	190
11.1.17.	SQLR0001 .....	190
11.2.	OS/390 .....	190
11.2.1.	RMF I .....	190
11.2.2.	RMF II .....	190
11.2.3.	RMF III .....	190
11.3.	DB2 .....	191
11.3.1.	DB2PM .....	191
12.	Appendix 2: Reference Materials .....	192
12.1.	SAP Manuals .....	192
12.2.	IBM manuals .....	192

## 1. Acknowledgements

First and foremost, 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 OS/390. I have added examples of solving specific problems, to demonstrate the process for starting from SAP performance indicators to drill-down to DB2 and OS/390 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 Lynn Nagel, Phil Hardy and Don Geissler who edited or reviewed the paper and made numerous suggestions for improvements. Thanks also to Mark Keimig, who showed me the process for SQL analysis with catalog statistics. Thank you to Walter Orb, who showed me how to interpret symptoms of many SAP and AIX performance problems.

## 2. Disclaimers

The paper contains examples from systems ranging from SAP 3.1I and DB2 V5 and OS/390 2.5 up to SAP 6.10, DB2 V7 and OS/390 2.9

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. Most examples have been taken verbatim from productive or stress test systems. A few have been edited to create clearer examples for the paper.

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.

## 3. Copyrights

SAP and R/3 are copyrights of SAP A.G.

DB2, Universal Database, MVS, OS/390, and z/OS are copyrights of IBM corp.

Oracle is a copyright of Oracle corp.

## 4. Feedback

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

## 5. 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 OS/390 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 OS/390, 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 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 SAPnotes. 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.

## 6. DB2/390 and SAP background information

The architecture and components of the connection between SAP and DB2 vary from release to release. There are several R/3 release-dependent “Planning Guides” that describe the architecture in detail. See section 12 for manual numbers and names.

Here are a few points that are important in understanding the way that SAP uses the DB2 database. The way that SAP uses DB2 is somewhat different than traditional DB2 applications.

### 6.1. SAP

- **SAP transaction, user, and security management** – end users send transactions to the application server. SAP manages the dispatch of transactions into SAP work processes. SAP does security checking from its data base tables to determine the rights of an SAP user.
- **SAP Unit-of-Work vs. DB2 Unit-of-Work (UOW)** – an SAP UOW, such as a transaction, can be made up of one or more SAP dialog steps. There is at least one DB2 UOW in each dialog step. In order to have transactional integrity across multiple DB2 UOWs, SAP has its own locking system, called “Enqueue”. SAP uses change queue tables (called VBLOG tables) to hold changes across multiple DB2 UOWs, until they are ready to be committed to application tables. As part of SAP UOW processing, subsequent SAP dialog steps for a transaction may add to the queued information in the VBLOG tables. At SAP commit time, all the changes for an SAP UOW are read from the VBLOG tables, and applied to the tables containing the business data.
- **Background job updates** – Background (SAP Batch) jobs may use the VBLOG tables, or may do updates directly into the business tables.
- **SAP caching** -- for performance reasons, SAP caches many SAP objects on the application servers. This includes SAP programs, transaction screens, and even some table data.
- **Commits required for read-only jobs** – SAP programs, which at the application level are read only, may be taking locks in the database during a statement prepare, program re-generation, etc. For this reason, it is important that all batch or long running programs issue periodic commits.
- **Referential integrity** – SAP manages all foreign key relationships and referential integrity constraints, though DB2 has the capability to do it.

### 6.2. DB2

- **DB2 security management** -- at the DB2 level, there is only one userid (by default SAPR3) that owns all objects in the SAP database, executes all SQL, etc. There is no way to use DB2 facilities to track object accesses back to a specific user in SAP.
- **Long running DB2 threads** -- SAP work processes connect to DB2 by sending a connect request to the ICLI (Integrated Command Level Interface). The ICLI can be thought of as a remote SQL server for SAP. When it receives the connection request, the ICLI creates a thread in DB2. Many SAP transactions, batch jobs, updates, etc., may be executed in each thread between thread creation and termination. Starting with SAP 4.5B, all threads use the same DB2 PLAN. (Previously, each thread had a uniquely named plan.) Thus, DB2 plan-based and thread-

based accounting cannot be easily used for analyzing performance of specific transactions or reports.

- **All dynamic SQL** -- SAP uses DB2 dynamic SQL, where the SQL is prepared and executed at program runtime, rather than SQL that is prepared (bound) before program runtime. Since prepared statements are prepared with parameter markers, the DB2 optimizer does not (by default) use some optimizer statistics, such as SYSCOLDIST data.
- **Execution time re-optimization** -- there are SAP programming hints which tell DB2 to re-optimize statements at execution time, when the variables are available. When statements are re-optimized at execution, DB2 can use all available optimizer statistics.
- **Thread local thread statement cache** – the SQL statements that a thread is executing are kept in a “local statement cache” in DBM1. Since each prepared statement takes on average 10KB, this local cache can cause a large demand for DBM1 VSTOR. It is common for large SAP systems to have only 400MB-600MB of bufferpools, with the rest of the 2GB of VSTOR in DBM1 used for thread local cache, EDM pool, etc. Hiperpools (or dataspace for systems with 64-bit real support) are used to make additional buffer memory available to DB2.
- **Table structure** – most tables have MANDT (client) as the leftmost column. In most systems, there is only one productive client, so MANDT has low cardinality, and will not filter rows well.



## 7. Solving a performance problem for a specific program or transaction

### 7.1. Check STAT 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 11.1.11, 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.

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

#### 7.1.1. Description of STAT record time components

Earlier 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	Roll time	Out	7 ms
Processing time	883 ms		In	0 ms
Load time	2 ms		Wait	0 ms
Generating time	0 ms			
Roll (in+wait) time	0 ms	Frontend	No.roundtrips	5
Database request time	38 ms		CUI time	863 ms
Enqueue time	0 ms		Net time	0 ms

**Figure 1: Sample STAT record**

Following is a summary of the components of response time. See SAPnote 8963 for a more detailed description for SAP releases up to 4.5B, SAPnote 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 SAPnote 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.
- **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 “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 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).

Since statistics counters can wrap, there can be delays that are not accounted for in the STAT records. When you are examining the STAT 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.
- 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 detailed database request time

In addition to the STAT overview shown in Figure 1, one can display detailed database request information in STAT. Depending on your release of SAP, this stanza will look like Figure 2, where database time per request is reported, or Figure 3, 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 | | Database | Requests | Database | Request | Avg.time |
| ABAP/4 request | Requests | rows | to buffer | calls | (time(ms) | 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 2: 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      |Database |      |Requests |Database | Request |Avg.time / |
| ABAP/4 request |  rows |Requests|to buffer |  calls |time (ms)| row (ms) |
-----
| Total        |    144 |    147 |    95 |    36 |    388 |    2.7 |
| Direct read  |     8  |    116 |    88 |     0 |    118 |   14.8 |
| Sequential read |   136 |     31 |     7 |     0 |    261 |     1.9 |
| Update       |     0  |     0  |     0 |     0 |     0  |     0.0 |
| Delete       |     0  |     0  |     0 |     0 |     0  |     0.0 |
| Insert       |     0  |     0  |     0 |     0 |     0  |     0.0 |
-----

```

Figure 3: 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 this is the case.* 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.

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”)
- Bad 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 8.3)
- Bad 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)
- Bad bufferpool hit rates or I/O constraint on DB server

High per-row insert times may be caused by

- Bad 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.

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)
-----
|          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 4: 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 5 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 | ABAP/IV Processor | Database Calls |
| Type     | Requests | Fails | Prepares | Opens | Fetch/Exec | Rows | Time [ms] |
|-----|-----|-----|-----|-----|-----|-----|-----|
| Select single | 0 | 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 5: rsdb/stattime time statistics in ST10**

## 7.2. **Actions from STAT 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 at other components of elapsed time for the cause.
  - If the performance of the other components of response time is 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 7.5.1.
  - Use ST06 (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 SAPnote 51373 and 161053 for ways to minimize impact of GUI RFC calls.



- 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 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 times, ST04 thread details, or DB2 trace with IFCID 44,45,226,227 (lock and latch suspension).
    - 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 somewhat important, but not 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 7.3.10.
- If GUI time is high
  - See SAPnotes 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.

### **7.3. Examples of problem indicators in STAT 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

### 7.3.1. Low CPU time example

Low CPU time can be an indicator of inefficient SQL. The database request time may be long, or in cases where the database request time statistics have wrapped, database request time may be short and processing time long.

If CPU time is a very small percentage of elapsed time, consider checking for inefficient SQL. Here CPU time is only 1% of elapsed time, which is generally a strong indicator of inefficient SQL. If the CPU time is <5%-10%, 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	Roll time	Out	0 ms
Processing time	325 ms		In	1 ms
Load time	5 ms		Wait	0 ms
Generating time	0 ms	Frontend	No.roundtrips	0
Roll (in+wait) time	1 ms		GUI time	0 ms
Database request time	19,870 ms		Net time	0 ms
Enqueue time	0 ms			

**Figure 6: STAT record with low CPU time**

Use ST05 to trace and explain slow SQL statements.

### 7.3.2. 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 11.1.1 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 7, 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	Roll time	Out	0 ms
Processing time	536,773 ms		In	1 ms
Load time	94 ms		Wait	0 ms
Generating time	0 ms	Frontend	No.roundtrips	0
Roll (in+wait) time	1 ms		GUI time	0 ms
Database request time	48,580 ms		Net time	0 ms
Enqueue time	971 ms			

Figure 7: STAT record with high CPU time

### 7.3.3. 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 8: STAT RFC detail

### 7.3.4. 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.

### 7.3.5. Wait for work process example

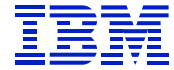
Wait for work process is often a symptom of another problem, where the root problem causes the dialog step to run slowly and keep the work process occupied for longer than optimal. This could be caused by CPU overload, OS paging, SAP buffer configuration, inefficient SQL, etc. Look at the components 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, “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 9: STAT wait for work process – symptom of SAP roll area overflow



In Figure 9, 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 10, 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      |           |Database | Requests |Database | Request |Avg.time |
| ABAP/4 request |Requests |  rows |to buffer |  calls |time(ms) |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 10: STAT slow insert causes wait time

### 7.3.6. Processing time examples

Processing time is a symptom of a problem when it is not consistent with other statistics. In the simplest case, such as Figure 11, where there are no RFC and GUI calls, processing time should be very close to CPU time. In Figure 11, 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	Roll time	Out	1801477 ms
Processing time	7,252 ms		In	6 ms
Load time	13 ms		Wait	48 ms
Generating time	0 ms	Frontend	No.roundtrips	0
Roll (in+wait) time	54 ms		GUI time	0 ms
Database request time	10,212 ms		Net time	0 ms
Enqueue time	94 ms			

**Figure 11: 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 12 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			
Response time	174,629 ms	Load time	Program	6 ms
			Screen	1 ms
			CUA interf.	2 ms
Wait for work process	0 ms	Roll time	Out	0 ms
Processing time	91,747 ms		In	1 ms
Load time	9 ms		Wait	0 ms
Generating time	0 ms			
Roll (int+wait) time	1 ms	Frontend	No.roundtrips	0
Database request time	82,872 ms		GUI time	0 ms
Enqueue time	0 ms		Net time	0 ms

Figure 12: Processing time shows missing time in SAP

Processing time should be evaluated against other information in STAT. In Figure 13, processing time is 3x CPU time, and so the program looks like it has a “missing time” problem. What has happened is that GUI time has not been included in Roll wait, and thus not removed from processing time. Here, there is not a “missing time” problem -- CPU + GUI + Enqueue is about the same as processing time.

Analysis of time in work process

CPU time	1,531 ms	Number	Roll ins	1
RFC+CPIC time	16 ms		Roll outs	1
			Enqueues	21
Response time	6,101 ms	Load time	Program	0 ms
			Screen	3 ms
			CUA interf.	0 ms
Wait for work process	0 ms	Roll time	Out	2 ms
Processing time	4,662 ms		In	2 ms
Load time	3 ms		Wait	0 ms
Generating time	0 ms			
Roll (int+wait) time	2 ms	Frontend	No.roundtrips	2
Database request time	1,378 ms		GUI time	3,160 ms
Enqueue time	56 ms		Net time	1,170 ms

Figure 13: Processing time containing GUI time

In Figure 14, GUI time is included in Roll wait, and so GUI time has been subtracted from processing time. Processing time agrees with CPU time, so there is no “missing time” problem.

Analysis of time in work process

CPU time	47 ms	Number	Roll ins	12
RFC+CPIC time	0 ms		Roll outs	12
			Enqueues	0
Response time	12,417 ms	Load time	Program	2 ms
			Screen	0 ms
Wait for work process	16 ms		CUA interf.	1 ms
Processing time	82 ms	Roll time	Out	3,565 ms
Load time	3 ms		In	4 ms
Generating time	0 ms		Wait	12,221 ms
Roll (in+wait) time	12,225 ms	Frontend	No. roundtrips	11
Database request time	91 ms		GUI time	12,221 ms
Enqueue time	0 ms		Net time	0 ms

Figure 14: processing time with GUI time removed

### 7.3.7. 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.

In Figure 15, 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-----14907321ms--| 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 15: STAT high load time

### 7.3.8. Roll (in+wait) time example

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

In Figure 16, 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 16: STAT roll (in+wait) GUI time

In Figure 17, roll (in+wait) points to overflow of the memory roll area on the application server, because there is no roll-wait time. Use ST02 to follow up and review the roll area memory usage.

```

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 17: STAT roll-in

### 7.3.9. Database Request time

Problems in database request time, caused by inefficient or slow SQL, are by far the most common performance problem in SAP. See section 7.4 and section 8.4 for examples of examining slow and inefficient SQL.

### 7.3.10. 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 1-5 ms per enqueue. If enqueue processing is a significant percentage of time in STAT, 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 (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.

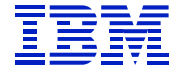
**The third type** of ENQ performance problems is an I/O bottleneck on the ENQBCK file. At the end of an SAP transaction, ABAP programs issue an SAP “commit work” command. The “commit work” signals that the transaction is finished, and its update can be processed. Because SAP uses enqueues to serialize 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.

### 7.3.10.1. Enqueue processor constraint example

The symptom of this problem is long ENQ times seen in ST03 or STAT/STAD. Additionally, when one checks ST06 “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
-----
176,228  |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 |
169,076 |prdadm      |dw.sapPRD_DVEBMGS00 |  2.18 |  2:50 | 42,912 | 61 |
198,834 |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 |
134,896 |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 18: ST06 > detail analysis > top CPU - showing processor constraint

**Important note:** Newer releases - such as 4.6D - 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.

```

-----
|No.Ty. PID      Status ReasonStart Err Sem CPU      Time  Program  ClieUser      Action      Table
-----
10 DIA 30112  running      Yes          82  SAPLEPTO 010 TRAIN001  Sequential read  BCONT
11 DIA 30222  running      Yes          81  SAPLEENM 010 SMITHAR   Roll In
12 DIA 30560  running      Yes          2135 SAPLSENA 010 WF-BATCH  Sequential read  TEWOCODE
13 BTC 31582  stopped ENQ  Yes          2147 SAPLSENA 010 BTCHUSER_ALL
14 BTC 30892  stopped ENQ  Yes          2144 SAPLSENA 010 BTCHUSER_ALL
15 DIA 31024  waiting      Yes          2144 SAPLSENA 010 BTCHUSER_ALL
16 BTC 31984  stopped ENQ  Yes  1          RSMON000 010 GORDONMR
17 DIA 25810  running      Yes
18 DIA 25576  waiting      Yes
19 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  ZCSVCO_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 19: SM50 showing ENQ wait

In Figure 19, 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 20: SM51 > Goto > queue information - display of queues on SAP central instance**

In Figure 20, 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.

**7.3.10.2. ENQBCK I/O constraint example**

The problem reported is slow batch performance. We look at STAT records to evaluate the components of elapsed time.



```

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)
-----
End time Tcod  Program  T Scr. Wp User      Response Memory  Wait    CPU    DB req.  Load/Gen kBytes  Phys. db
time(ms) used(kB) time(ms) time(ms) time(ms) time(ms) transfer changes
-----
|20:44:44      RM07II35 B      33 ST0_PI_BATCH|5853955 | 6,346 | 0 |976,950 |1394897 | 21 |284478.2| 3,730 |
|20:48:00      RM07II35 B      32 ST0_PI_BATCH|6051193 | 6,347 | 0 |1000320 |1411926 | 22 |291504.2| 3,888 |
|20:51:44      RM07II35 B      34 ST0_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. Enqueues should normally take just a few ms each, this is very unusual.

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 ClieUser      Action      Table
-----
|0 DIA 88764     running Yes      26      22   SAPLSENT 010 ST0_PI_BATCH
|1 DIA 79664     waiting Yes
|2 DIA 74754     running Yes
|3 DIA 70410     waiting Yes
|4 DIA 59196     waiting Yes
|5 DIA 27454     waiting Yes
|6 BTC 91694     stopped UPD Yes      1209  RM07II35 010 ST0_PI_BATCH
|7 BTC 113028    running Yes      26268 ZRM07MAD 010 ST0_ARCH  Delete     MSBG
|8 BTC 44674     running Yes      215   SAPLSAL2 010 ST0_EXE
|9 BTC 57852     waiting Yes
|10 DIA 96054    running Yes      26      10      010 ST0_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 ST0_OPC_CPIC
|14 DIA 50312    running Yes      26      22   SAPLSENT 010 ST0_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      Resident Prior.
      [%]      [s]      size [kB]
-----
|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      |gwrld -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.

Since I/O on enqbk 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 24: 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 25: 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.

### 7.3.11. Frontend example

With the new GUI design in 4.6, 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 new 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 | Roll time    Out         4 ms |
| Load time                6 ms      In           2 ms |
| Generating time          0 ms      Wait         543 ms |
| Roll (in+wait) time      545 ms | Frontend     No.roundtrips 3      |
| Database request time    94 ms      GUI time     3,780 ms |
| Enqueue time             0 ms      Net time     1,212 ms |
|                          |                          |                          |
-----

```

Figure 26: STAT with long GUI time

In this example in Figure 26, 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.

SAPnotes 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.

### 7.3.12. Missing time in STAT – 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 or an OS program such as `vmstat` to check for paging.
- **There is a CPU overload on the application server.** Use ST06 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 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.

## 7.4. Transaction

### 7.4.1. Analysis process for transactions

In the case where there is a performance problem with a specific 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.

### 7.4.2. Sample transaction analysis - MIRO

In this case, we are investigating a performance problem that has been reported with the transaction MIRO. We ask a user who is experienced in running the transaction and who has reported performance problems to run the transaction while we trace it.

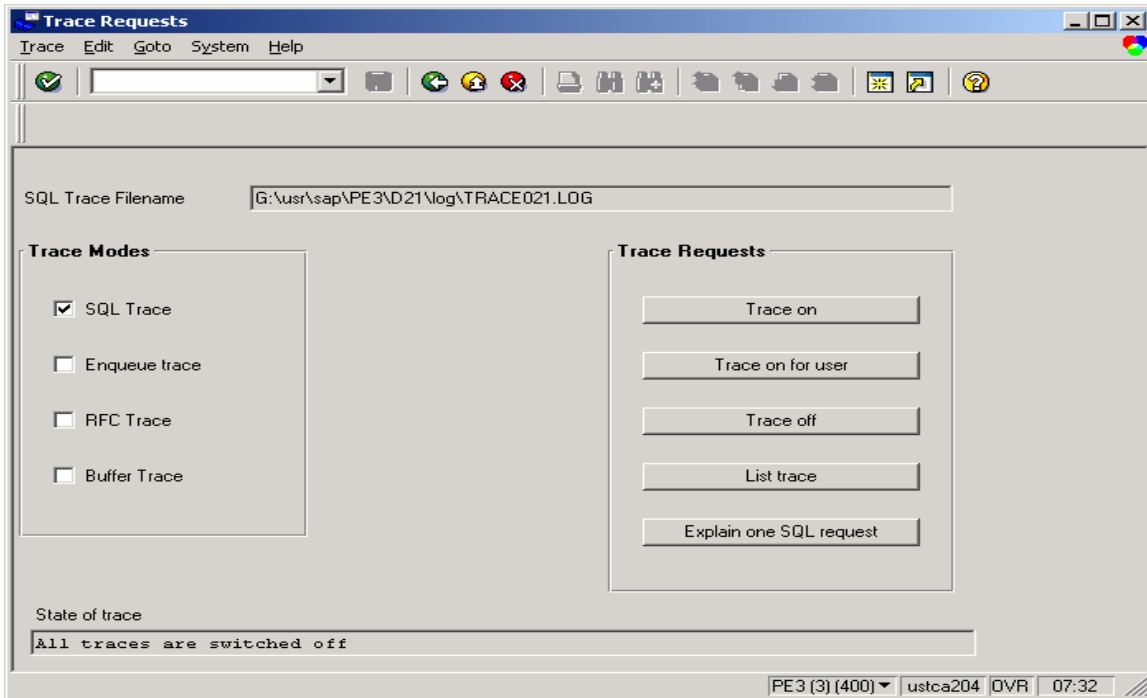
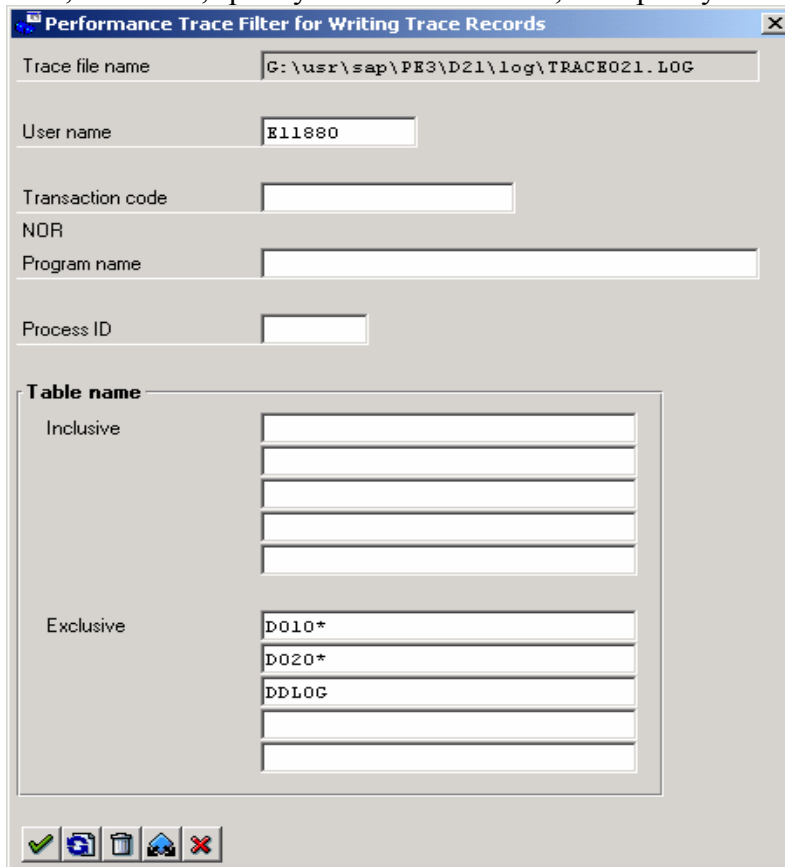


Figure 27: MIRO ST05

First, run ST05, specify “trace on for user”, and specify user name.



Performance Trace Filter for Writing Trace Records

Trace file name: G:\usr\sap\PE3\D21\log\TRACE021.LOG

User name: E11880

Transaction code:

NOR

Program name:

Process ID:

**Table name**

Inclusive

Exclusive

D010\*  
D020\*  
DDL0G

OK Cancel Help Close

**Figure 28: ST05 selection screen**

Note in Figure 28, one can narrow the performance trace by transaction, program, table, etc. The ST05 trace can generate a large trace file. If we have already narrowed the problem down to a few tables and need to reduce the size of the trace file to run the trace longer, the filters can be used to select a specific table or set of tables.

The SAP parameter *rstr/max\_diskpace* can be used to enlarge the size of the trace file for ST05. See SAPnote 25099 for the process to change the size of the trace file.

After the test is finished, turn the trace off (ST05 > trace off)

Then list the trace (ST05 > list trace).

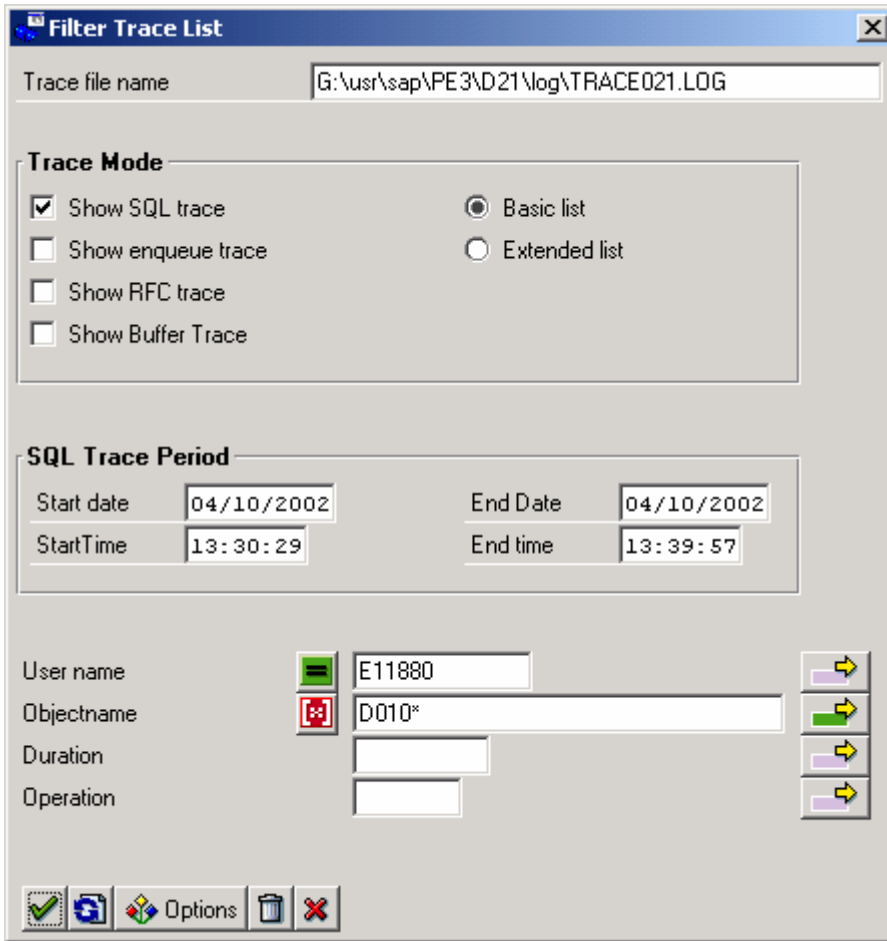


Figure 29: MIRO ST05 trace list selection screen

After a trace has been done, one can specify selection criteria, to further filter the list.

Duration	ObjectName	Op.	Rec	RC	Statement
787	BSIP	REOPEN	0	0	SELECT WHERE "MANDT" = '400' AND "LIFNR" = '0010006171' AND "WAERS" =
17,723	BSIP	FETCH	0	- 514	
15,137	BSIP	PREPARE	0	0	SELECT WHERE "MANDT" = ? AND "LIFNR" = ? AND "WAERS" = ? AND "WRBTR"
6	BSIP	EXECSTA	0	0	REOPEN
2,289	BSIP	FETCH	0	0	
29	REKP	REOPEN	0	0	SELECT WHERE "MANDT" = '400' AND "LIFNR" = '0010006171' AND "WAERS" =
1,207	REKP	FETCH	0	- 514	
8,026	REKP	PREPARE	0	0	SELECT WHERE "MANDT" = ? AND "LIFNR" = ? AND "WAERS" = ? AND "RMWWR"
6	REKP	EXECSTA	0	0	REOPEN
172,156	REKP	FETCH	0	0	
21	NRIV	REOPEN	0	0	SELECT WHERE "CLIENT" = '400' AND "OBJECT" = 'RE_BELEG' AND "SUBOBJEC"
1,060	NRIV	FETCH	0	- 514	
1,448	NRIV	PREPARE	0	0	SELECT WHERE "CLIENT" = ? AND "OBJECT" = ? AND "SUBOBJECT" = ? AND "
6	NRIV	EXECSTA	0	0	REOPEN
1,177	NRIV	FETCH	1	0	
2,148		EXECSTA	0	0	COMMIT
34	BSIP	REOPEN	0	0	SELECT WHERE "MANDT" = '400' AND "LIFNR" = '0010006171' AND "WAERS" =
2,014	BSIP	FETCH	0	0	
23	REKP	REOPEN	0	0	SELECT WHERE "MANDT" = '400' AND "LIFNR" = '0010006171' AND "WAERS" =
144,679	REKP	FETCH	0	0	

Figure 30: ST05 > list trace - slow RBKP

Figure 30 shows selects against RBKP that are slow – about 150ms – and that return no rows. *ST05 Duration is in micro-seconds, and the “Rec” column shows rows returned for the FETCH.*

In the list above, select the prepare statement and press explain.

```

SELECT * FROM "REKP" WHERE "MANDT" = ? AND "LIFNR" = ? AND "WAERS" = ? AND "RMWWR" = ? AND "BURRS" = ? A

Explanation of query block number: 1 step: 1
Query block type is SELECT
Performance is good
Index is used. Index scan by matching index.

Method: access new table.
data pages are read in advance
prefetch through a page list
new Table: SAPR3.REKP
table space locked in mode: N
Accesstype: by index.
Index: SAPR3.REKP-3 (matching Index)
Index columns (ordered): MANDT
USNAM
RESTAT
IVTYP
with 1 matching columns of 4 Index-Columns.
    
```

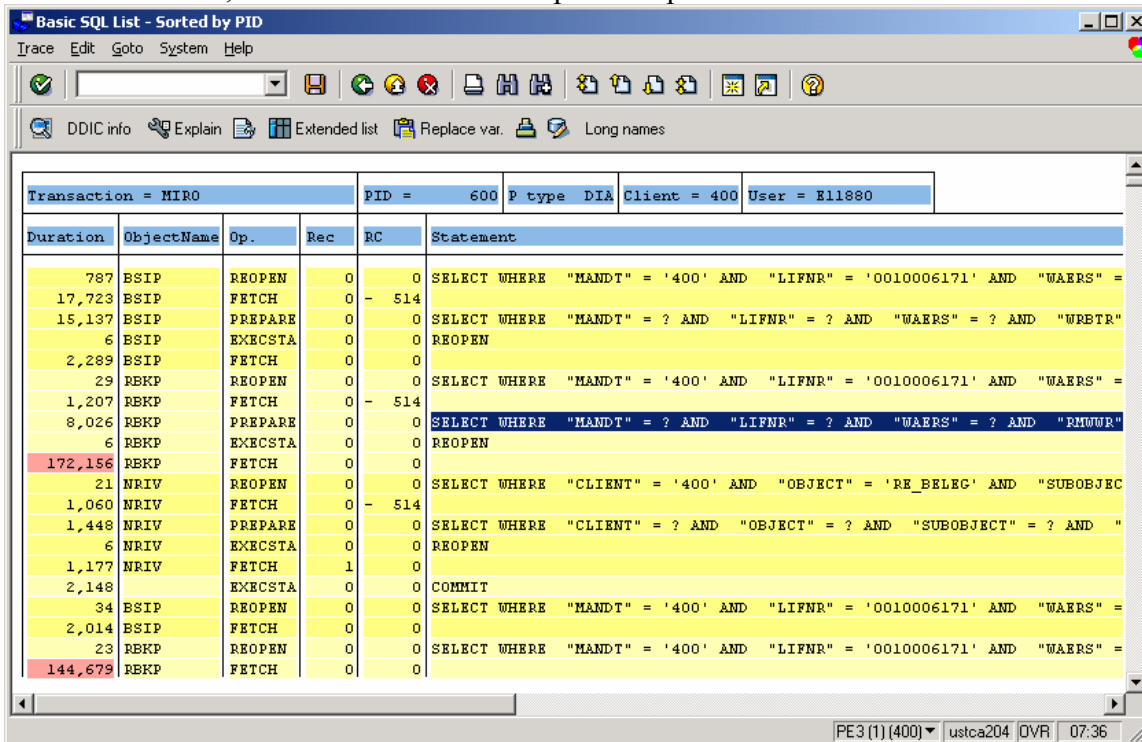
Figure 31: ST05 RBKP Explain

At the top of Figure 31, it says “performance is good”. When reviewing the explain output in SAP, take this comment with a grain of salt. It means that an index is being used. As this example shows, even though an index is being used, the performance is not good. Non-matching index scan, where the entire index may be read, will also say “performance is good”, and tablespace scan, which may be the right choice when many rows will be read, will say “performance is bad”.

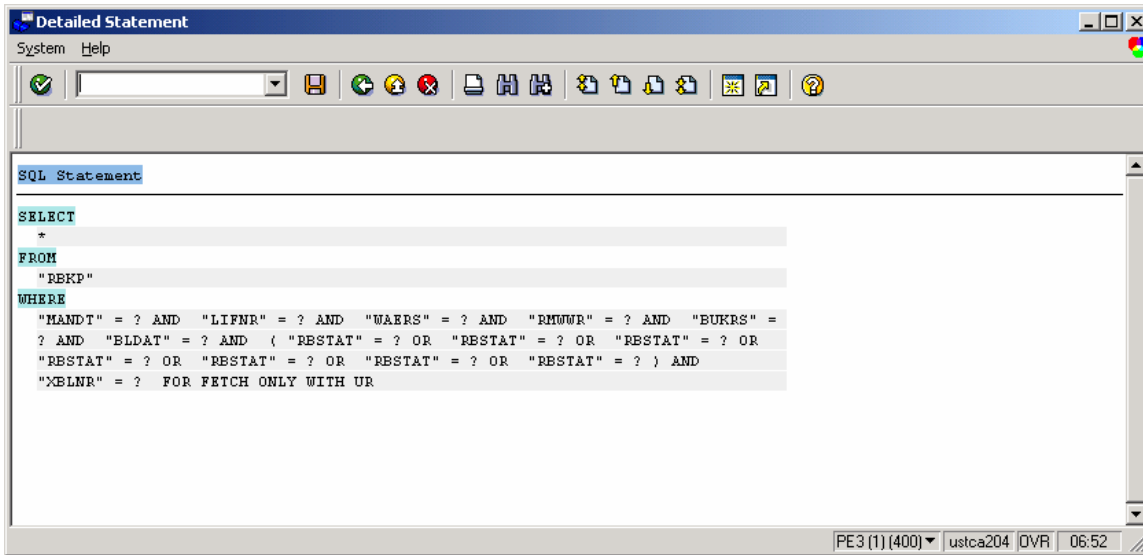
The explain output in Figure 31 shows that only the MANDT column is matched in the index. Since there is generally only one value in MANDT, this does not help to select the result rows. Even if there are several different values in MANDT, the data will generally be skewed such that almost all the rows are in one MANDT (the productive MANDT) so filtering on MANDT is not helpful.

Next check the columns in the predicates to see if DB2 might be doing index screening, which would be better than matching only MANDT. Predicates are the “column operation var” selection criteria in the SQL. DB2 does index screening when there are gaps in the index columns referenced by the predicates, e.g. the predicate uses columns one and three in an index with three columns. In order to confirm index screening we need to check the SQL.

In the ST05 trace, select the statement and press “replace vars” or drill into the REOPEN statement.



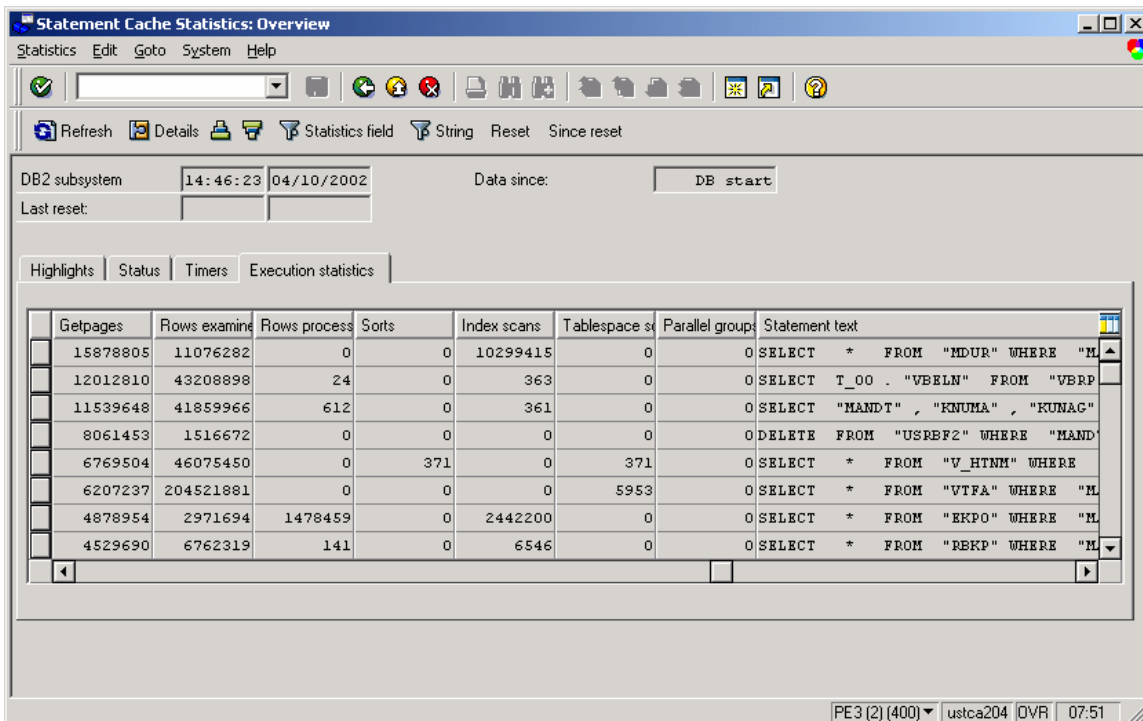
Duration	ObjectName	Op.	Rec	RC	Statement
787	BSIP	REOPEN	0	0	SELECT WHERE "MANDT" = '400' AND "LIFNR" = '0010006171' AND "WAERS" =
17,723	BSIP	FETCH	0	- 514	
15,137	BSIP	PREPARE	0	0	SELECT WHERE "MANDT" = ? AND "LIFNR" = ? AND "WAERS" = ? AND "WBETR"
6	BSIP	EXECSTA	0	0	REOPEN
2,289	BSIP	FETCH	0	0	
29	REKP	REOPEN	0	0	SELECT WHERE "MANDT" = '400' AND "LIFNR" = '0010006171' AND "WAERS" =
1,207	REKP	FETCH	0	- 514	
8,026	REKP	PREPARE	0	0	SELECT WHERE "MANDT" = ? AND "LIFNR" = ? AND "WAERS" = ? AND "RMUUR"
6	REKP	EXECSTA	0	0	REOPEN
172,156	REKP	FETCH	0	0	
21	NRIV	REOPEN	0	0	SELECT WHERE "CLIENT" = '400' AND "OBJECT" = 'RE_BELEG' AND "SUBOBJEC
1,060	NRIV	FETCH	0	- 514	
1,448	NRIV	PREPARE	0	0	SELECT WHERE "CLIENT" = ? AND "OBJECT" = ? AND "SUBOBJECT" = ? AND "
6	NRIV	EXECSTA	0	0	REOPEN
1,177	NRIV	FETCH	1	0	
2,148		EXECSTA	0	0	COMMIT
34	BSIP	REOPEN	0	0	SELECT WHERE "MANDT" = '400' AND "LIFNR" = '0010006171' AND "WAERS" =
2,014	BSIP	FETCH	0	0	
23	REKP	REOPEN	0	0	SELECT WHERE "MANDT" = '400' AND "LIFNR" = '0010006171' AND "WAERS" =
144,679	REKP	FETCH	0	0	



**Figure 32: Statement text with parameter markers – ST05 “replace vars”**

The columns in the statement are MANDT, LIFNR, WAERS, RMWWR, BUKRS, BLDAT, RBSTAT, and XBLNR. Comparing the SQL statement against index RBKP~3 that is being used, RBSTAT also matches, so DB2 may be using index screening on RBSTAT.

Look for the statement in ST04 (*the last line*), to get an idea of what the statement is doing in the 150 ms that it takes to execute. ST04 statement statistics will be discussed further in section 8.3.



**Figure 33: ST04 cached statement statistics with RBKP statement**

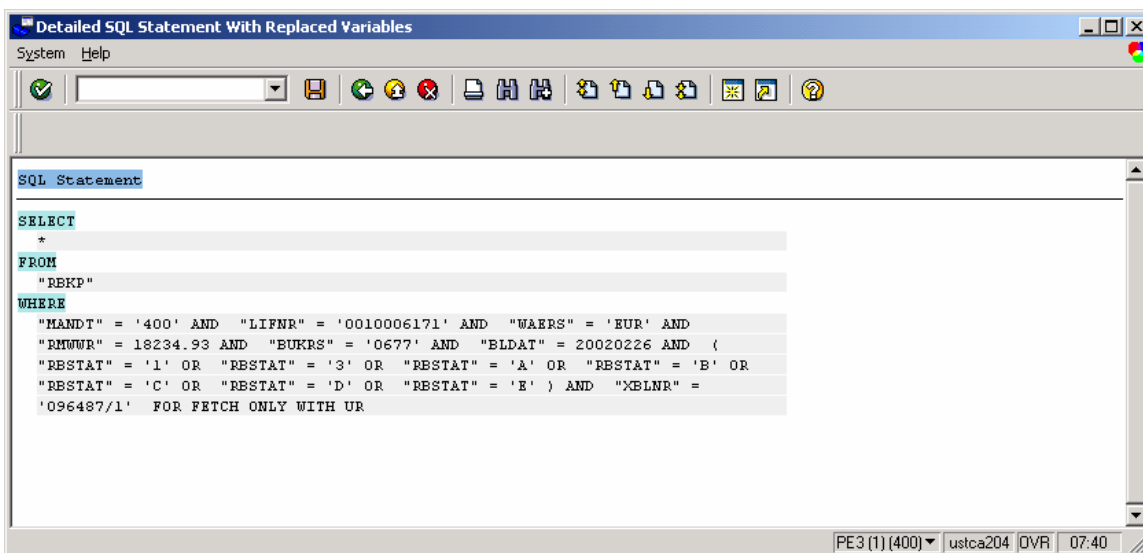
In Figure 33, last line, note that the statement did 4.5M getpages, examined 6.7M rows, and returned 141 rows. *In the ST04 cache statistics, a “row examined” means that DB2 had to look at the table row*



to determine whether the row satisfied the predicates, or to return the result row. This statement is probably not index screening, since the symptom of index screening is high getpages per row processed, but low rows examined per row process. In this case, DB2 is going to the table to evaluate the predicates.

Note in Figure 32 that the variables were not filled out when prepare is done, since dynamic SQL statements are prepared with parameter markers, and the variables are filled in at statement execution.

To see the parameter values for the SQL statement, select the REOPEN statement and press “replace var”. Checking the parameter values can be useful when examining statements that have predicates on low cardinality columns, such as status columns. By examining the variables, you can determine if the statement is searching for a value that seldom occurs, where matching the index could quickly return the rows.

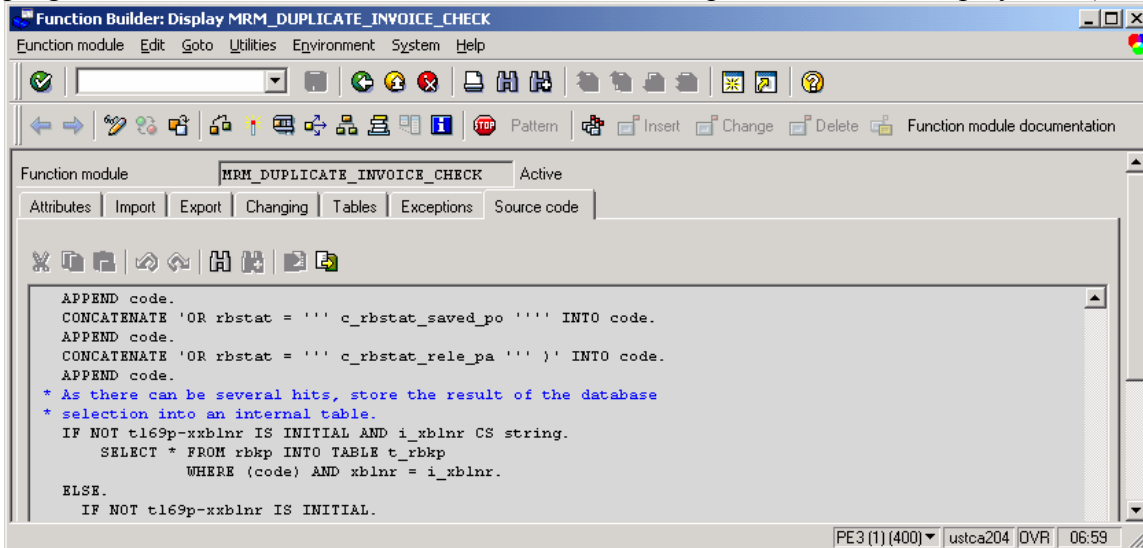


```
SQL Statement

SELECT
*
FROM
"REKP"
WHERE
"MANDT" = '400' AND "LIFNR" = '0010006171' AND "WAERS" = 'EUR' AND
"RMWUR" = 18234.93 AND "EUKRS" = '0677' AND "BLDAT" = 20020226 AND (
"RSTAT" = '1' OR "RSTAT" = '3' OR "RSTAT" = 'A' OR "RSTAT" = 'B' OR
"RSTAT" = 'C' OR "RSTAT" = 'D' OR "RSTAT" = 'E' ) AND "XBLNR" =
'096487/1' FOR FETCH ONLY WITH UR
```

Figure 34: ST05 display of SQL statement with parameter values

Now that we have determined that an inefficient access path is being used (each statement takes 150ms, and many rows were examined to return few rows), we look at the ABAP source code for the executing program. Select the REOPEN or PREPARE lines and press the ABAP display icon (the paper).



**Figure 35: ST05 source display of RBKP select with SAP dynamic SQL**

The function module (MRM\_...) is not in the customer name space -- this looks like SAP code. *Customer written code is Z\* and Y\* programs, as well as SAPxY\* and SAPxZ\* (e.g. SAPFY\*, SAPLY\*, SAPMZ\*), which contain user exits, function modules, etc.*

Note the line “WHERE (code)” in Figure 35. This is SAP dynamic SQL, where the statement is built at runtime by the program. This is different from DB2 dynamic SQL, which is statement preparation at runtime. SAP dynamic SQL such as these are very difficult to locate with the SE11 “where used”, since the predicates are not in the program source. Since the predicates are not in the source, we cannot use filters such as column name to find SQL statements in the “where used” list. See section 8.4.1 for an example of SE11 “where used”.

Use ST04 to check DB2 catalog statistics to find the cardinality of the columns specified in the SQL, to see if any would make good candidates for accessing the table. The catalog browser (ST04 > DB2 catalog browser) is not available with all versions of SAP. If your version of SAP does not have the catalog browser in ST04, these queries can be entered via SPUFI.

When reviewing catalog statistics, check the date on which runstats was run on the object. This can be done via DB02 table “detail analysis” or in ST04 catalog browser or SPUFI. Use “select \*”, rather than specifying the column names as in the examples below. The date will be contained with the other catalog statistics

The ST04 catalog browser does not come with pre-defined queries. The queries in this paper are shown as templates that you can enter in ST04.

Cardinality is the number of unique values in a column. A high cardinality column (or index) will filter the result set well, since there will be fewer rows in the table for each unique value in the index. Given the choice between a high cardinality index, and low cardinality index, DB2 will choose the index with high cardinality, since this should help to eliminate more rows at the time the index is read. If DB2 chose the low cardinality index, it would then have to eliminate many rows by examining the rows in the table, too.

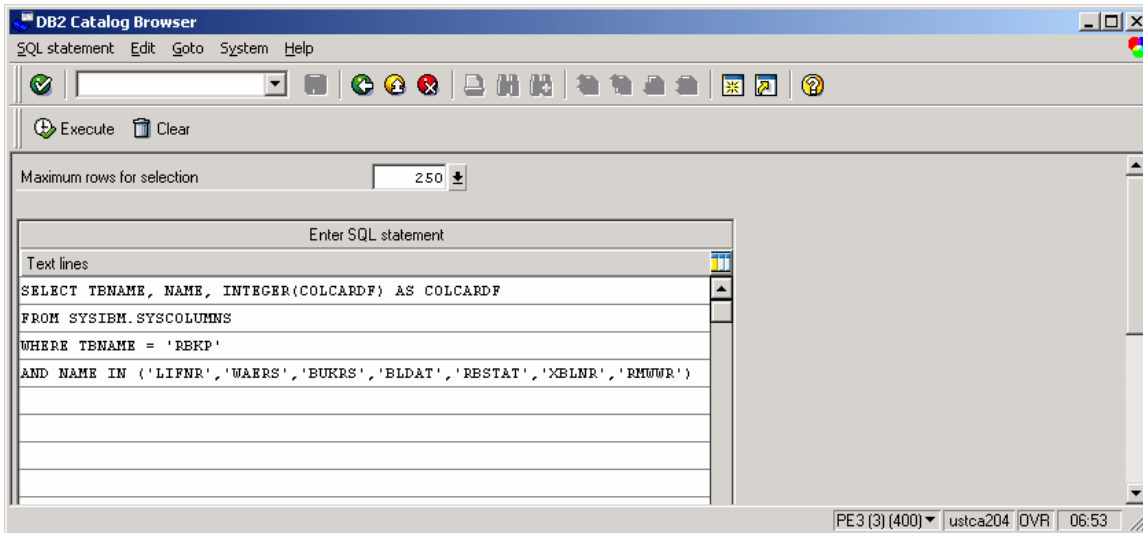


Figure 36: ST04 DB2 catalog browser to query catalog statistics

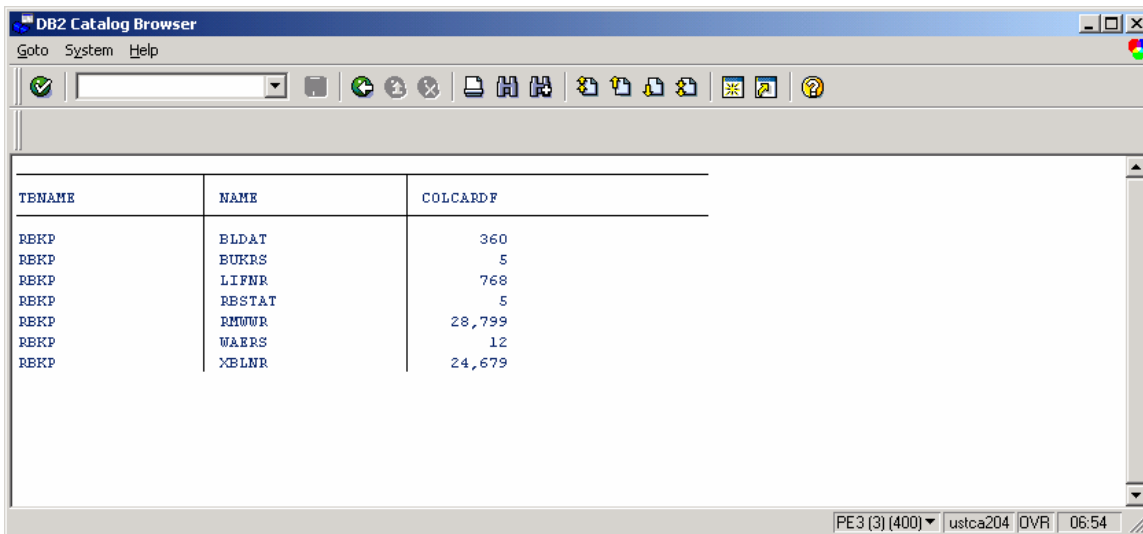


Figure 37: MIRO example - check column cardinality via catalog browser

Yes, XBLNR and RMWWR both have high cardinality, and are specified in the SQL, so if an index on one of these two columns were available, it would likely be better to use it.

Using ST04 DB2 catalog browser, check the indexes on the table, to see if there are any that match the columns, but were not used for some reason.

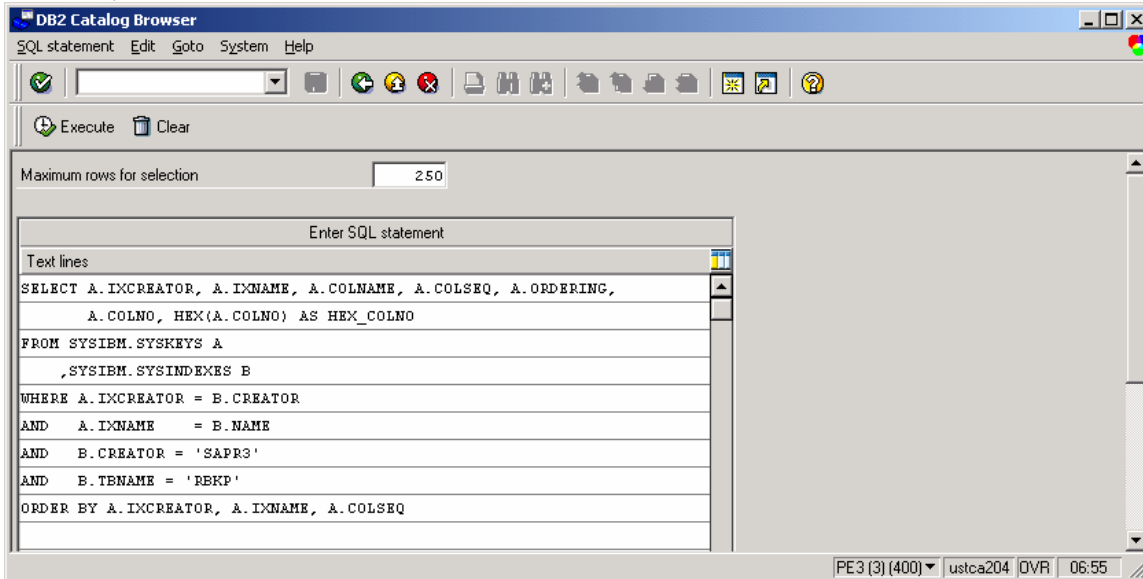


Figure 38: MIRO example - query to display indexes on RBKP

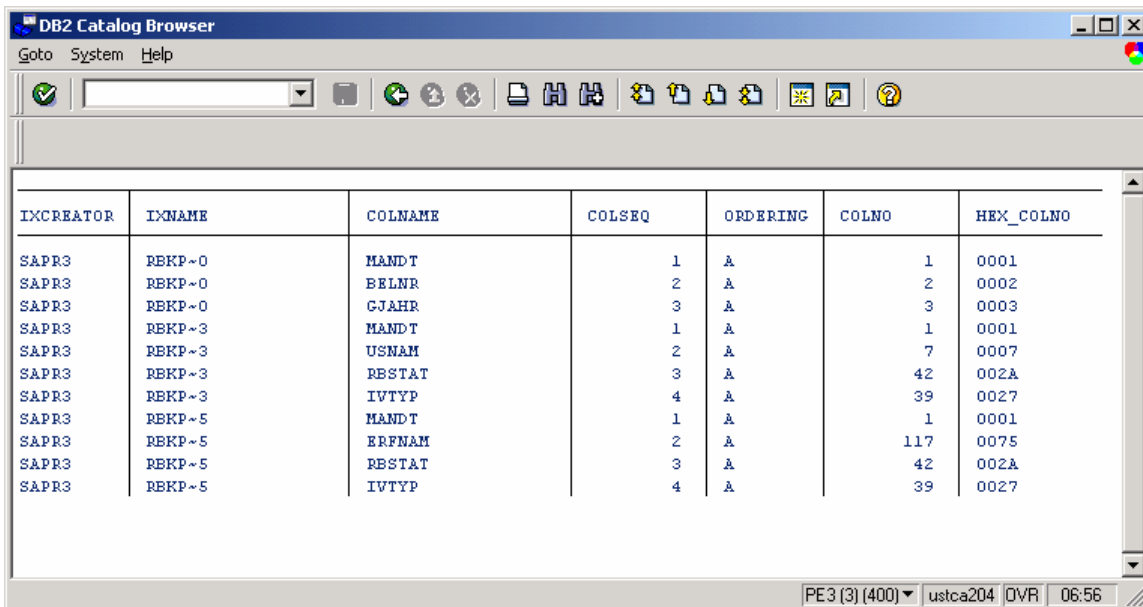


Figure 39: MIRO example - columns in indexes on RBKP

XBLNR and RMWWR do not appear in any indexes, so there are no indexes on the database that match the high cardinality columns in the SQL.

While we don't need to check the other indexes, since they are not used, if we were to check index cardinality, we would use the following command.

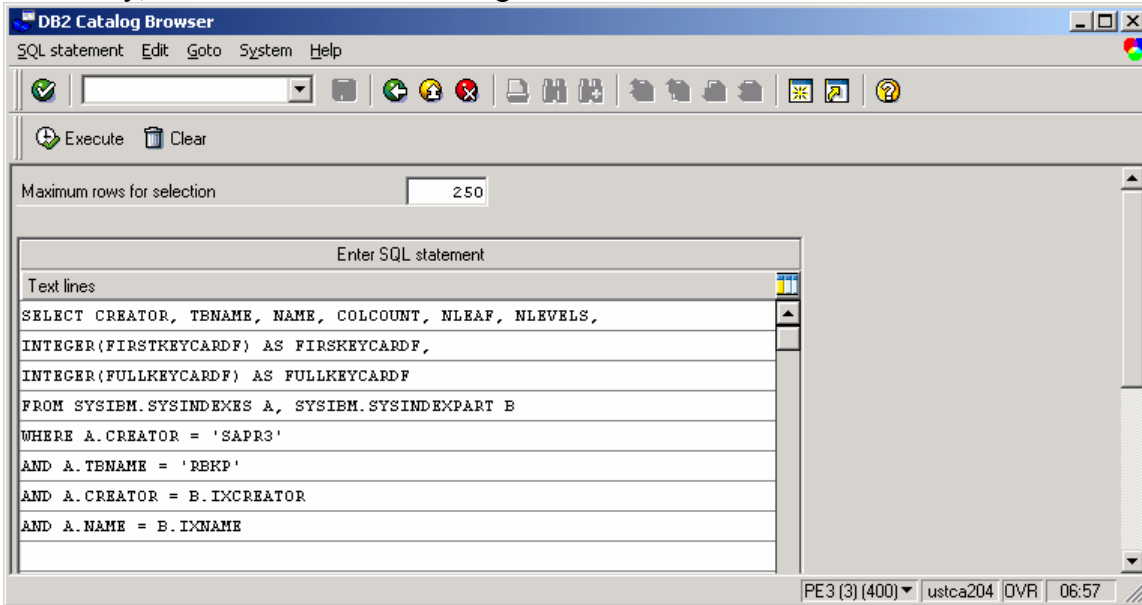


Figure 40: MIRO example - query to check index cardinality

The information on index cardinality (FULLKEYCARDF) is:

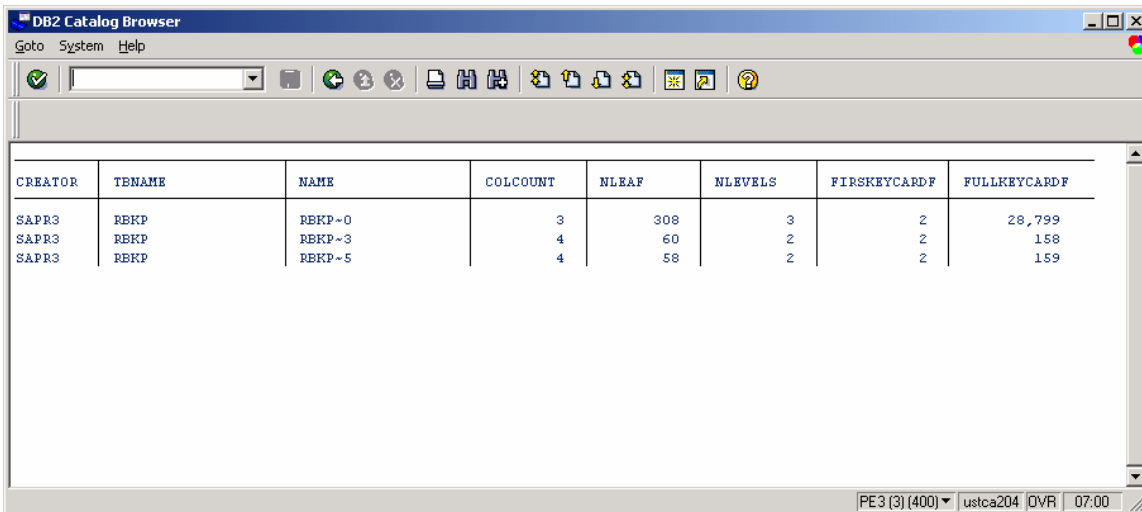
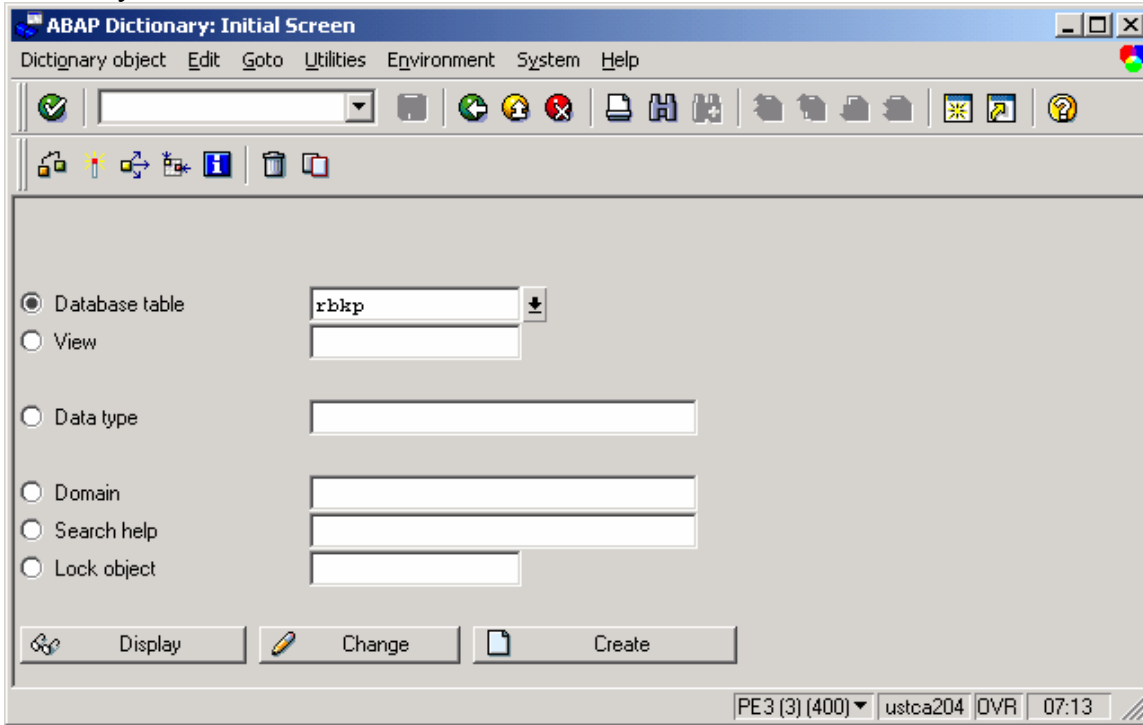
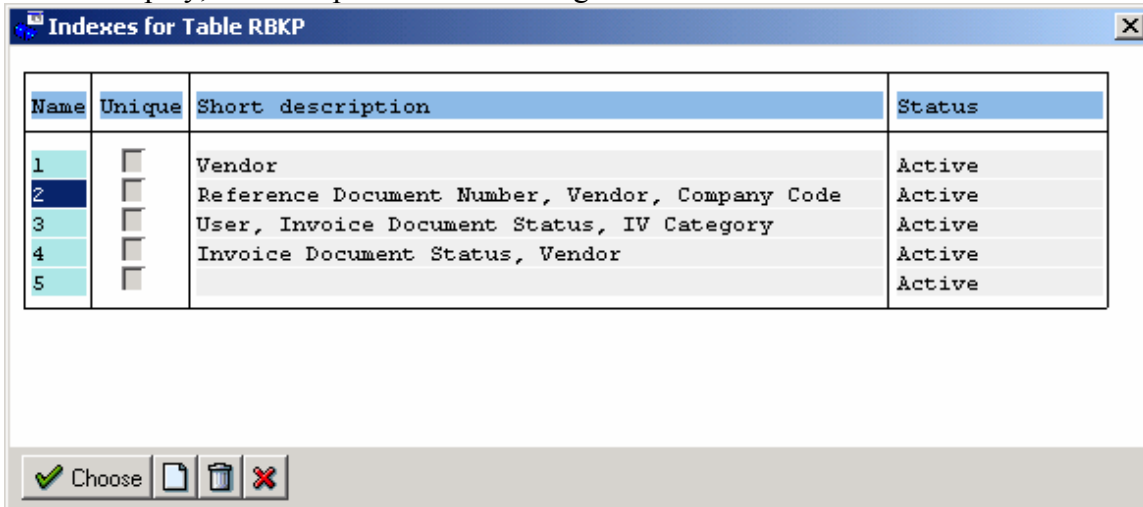


Figure 41: RBKP indexes and cardinality

Now, since it is SAP code, check SE11, to determine whether there is an index defined in the data dictionary that is not active on the database.

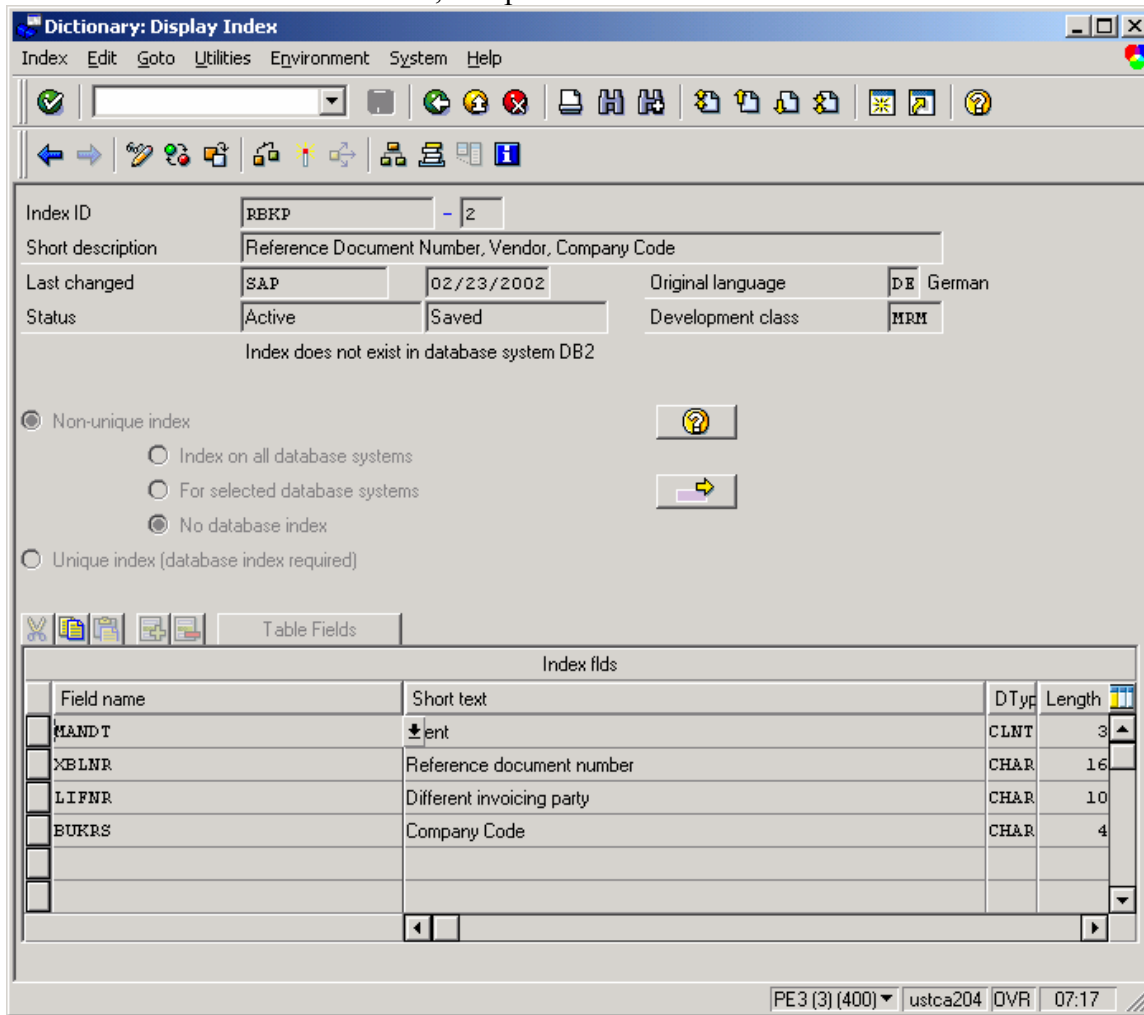


SE11 > display, and then press “indexes” to get this list.



It can be seen that there are 5 secondary indexes in the data dictionary (making 6 total), though only three indexes were defined on the database in Figure 41. SE11 > indexes displays only the secondary indexes. One can use SE14 to list all indexes, including the primary index

Select the second index in the list, and press choose.



**Figure 42: SE11 display index definition**

XBLNR is the second column, so this data dictionary definition is a match for the SQL statement . XBLNR had high cardinality. This index needs to be activated on the database. Note that status “active” in Figure 42 means that the data dictionary definition of the index is active, not that the index exists. Whether the index exists or not is displayed below the status.

Since the problem was found in SAP code, the action plan was to first check SE11 for defined but inactive indexes. If an index is not found in SE11, it is suggested that a search for a SAPnote about this problem should be performed. If no SAPnote is found, then opening an OSS message regarding the problem might be appropriate, or just create an index.

### 7.4.3. Sample transaction analysis – ME21L

Transaction ME21L has been reported to be slow. Checking the STAT records shows that there is one dialog step that seems to have inefficient SQL – nearly 30 ms per select, and almost 10 ms per row (3606 ms for select / 496 rows = 7 ms per row). This is rather slow. Since “database rows” is much greater than “requests”, we use the per-row times to evaluate performance.

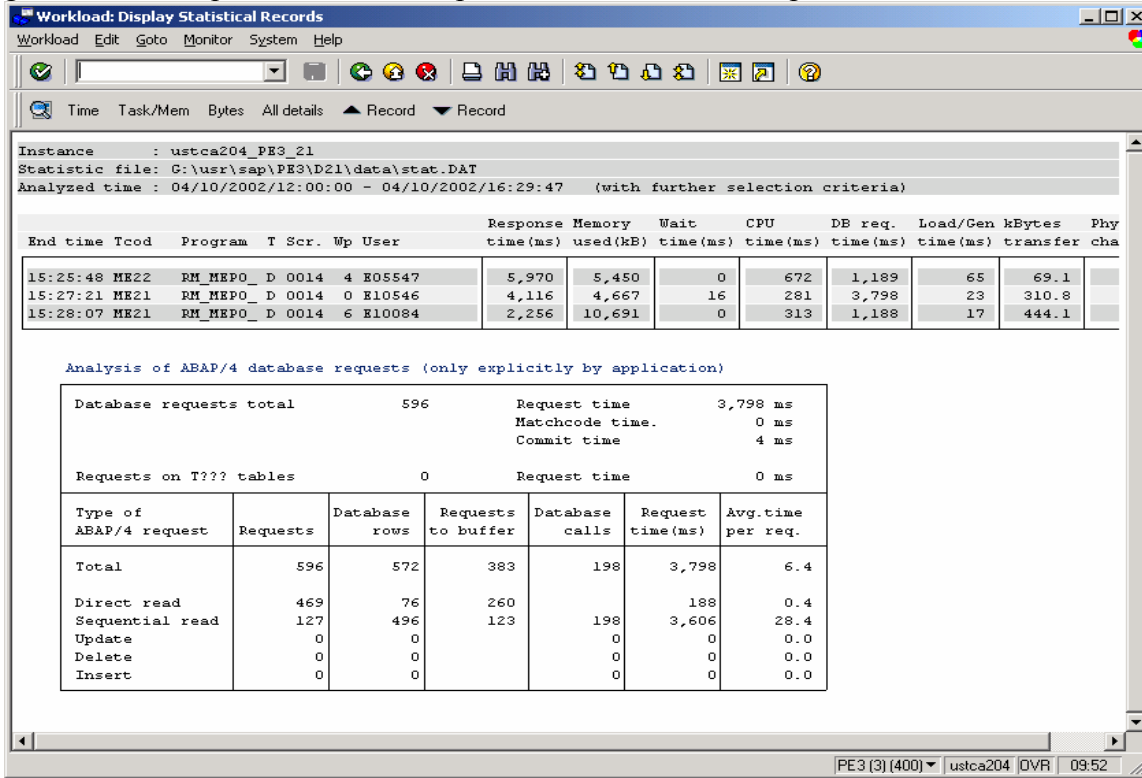


Figure 43: STAT record for ME21



Next, call our user who reported the problem and run ST05 trace while they execute the transaction. After transaction is traced, list and summarize the trace.

PID	Typ	Cli	Time	Tcode/prog	Table	SQL op	Recs.	Time
1752	DIA	400	16:27:09.514	ME21	BDSPHI02	SEL	1	1,679
1752	DIA	400	16:27:09.516	ME21	BDSPHF2	SEL	1	1,639
1752	DIA	400	16:27:09.521	ME21	BDSPHI02	SEL	1	1,805
1752	DIA	400	16:27:09.523	ME21	BDSPHF2	SEL	1	3,084
1752	DIA	400	16:27:09.557	ME21	- COMMIT WORK -	CW	0	2,377 C
1752	DIA	400	16:27:15.838	ME21	ADRC	SEL	1	2,681
1752	DIA	400	16:27:15.841	ME21	ADRC	SEL	1	15,009
1752	DIA	400	16:27:15.857	ME21	ADRC	SEL	0	1,196
1752	DIA	400	16:27:16.307	ME21	KLAH	SEL	3	3,260
1752	DIA	400	16:27:16.316	ME21	KSML	SEL	1	2,554
1752	DIA	400	16:27:16.321	ME21	CABN	SEL	1	35,512
1752	DIA	400	16:27:16.366	ME21	CAWN	SEL	0	1,438
1752	DIA	400	16:27:16.371	ME21	KSSK	SEL	0	1,972
1752	DIA	400	16:27:16.388	ME21	KSSK	SEL	1	2,897,927
1752	DIA	400	16:27:19.287	ME21	AUSP	SEL	0	2,019
1752	DIA	400	16:27:19.290	ME21	KSSK	SEL	0	2,362
1752	DIA	400	16:27:19.305	ME21	NRIV	SEL	1	2,655
1752	DIA	400	16:27:19.308	ME21	NRIV	UPD	1	1,551
1752	DIA	400	16:27:19.892	ME21	ESDUS	UPD	1	30,892
1752	DIA	400	16:27:19.934	ME21	ESDUS	UPD	1	17,964
1752	DIA	400	16:27:20.020	ME21	- COMMIT WORK -	CW	0	19,234 C
1752	DIA	400	16:28:50.786	ME21	ESDUS	SEL	103	101,212

Figure 44: Summarized ST05 SQL trace with slow KSSK

Figure 44 shows that KSSK looks like the problem. It takes almost three seconds to return one row. (The ST05 trace summary time units are microseconds.) Go back to the ST05 list to explain the statement. Select the reopen line, and press explain.

Duration	ObjectName	Op.	Rec	RC	Statement
24	KSSK	REOPEN	0	0	SELECT WHERE "MANDT" = '400' AND "OBJEK" = '0000000360' AND "MA
1,948	KSSK	FETCH	0	0	
22	KSSK	REOPEN	0	0	SELECT WHERE "MANDT" = '400' AND "CLINT" IN ( 0000000360 , 00000
2,897,905	KSSK	FETCH	1	0	
25	AUSP	REOPEN	0	0	SELECT WHERE "MANDT" = '400' AND "OBJEK" IN ( '01PO' , '01PO' ,
1,994	AUSP	FETCH	0	0	
18	KSSK	REOPEN	0	0	SELECT WHERE "MANDT" = '400' AND "CLINT" = 0000000360 AND "MAFI
2,344	KSSK	FETCH	0	0	
19	NRIV	REOPEN	0	0	SELECT WHERE "CLIENT" = '400' AND "OBJECT" = 'RINKBELEG' AND "S
2,636	NRIV	FETCH	1	0	

Figure 45: ST05 trace list with slow KSSK

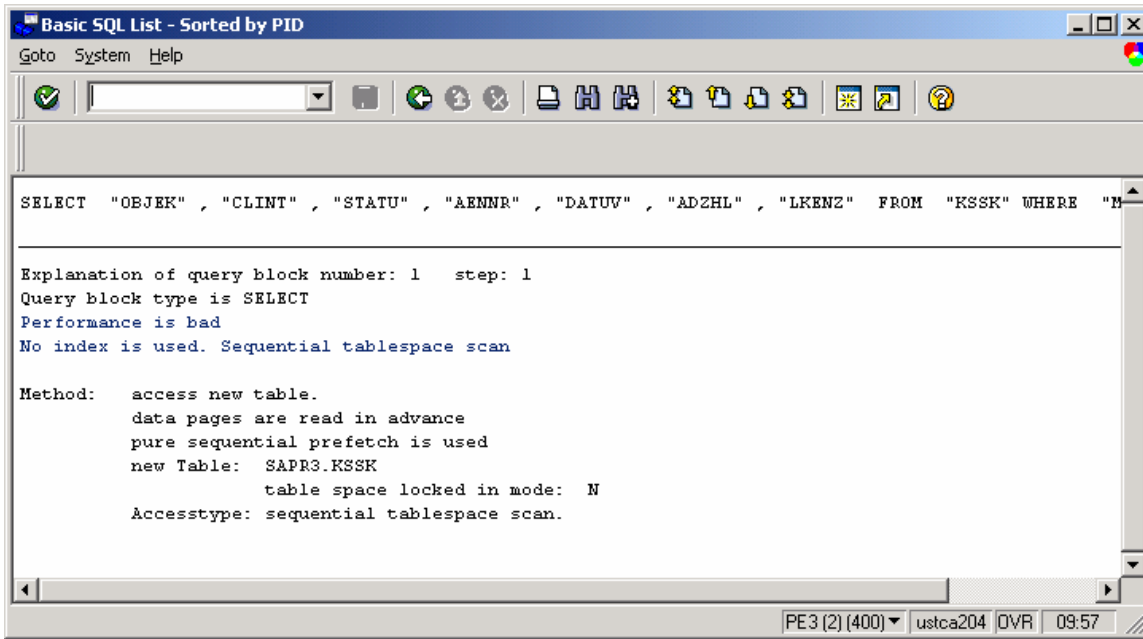


Figure 46: ST05 explained KSSK statement

It does not look good – the entire table is scanned. No index was used.

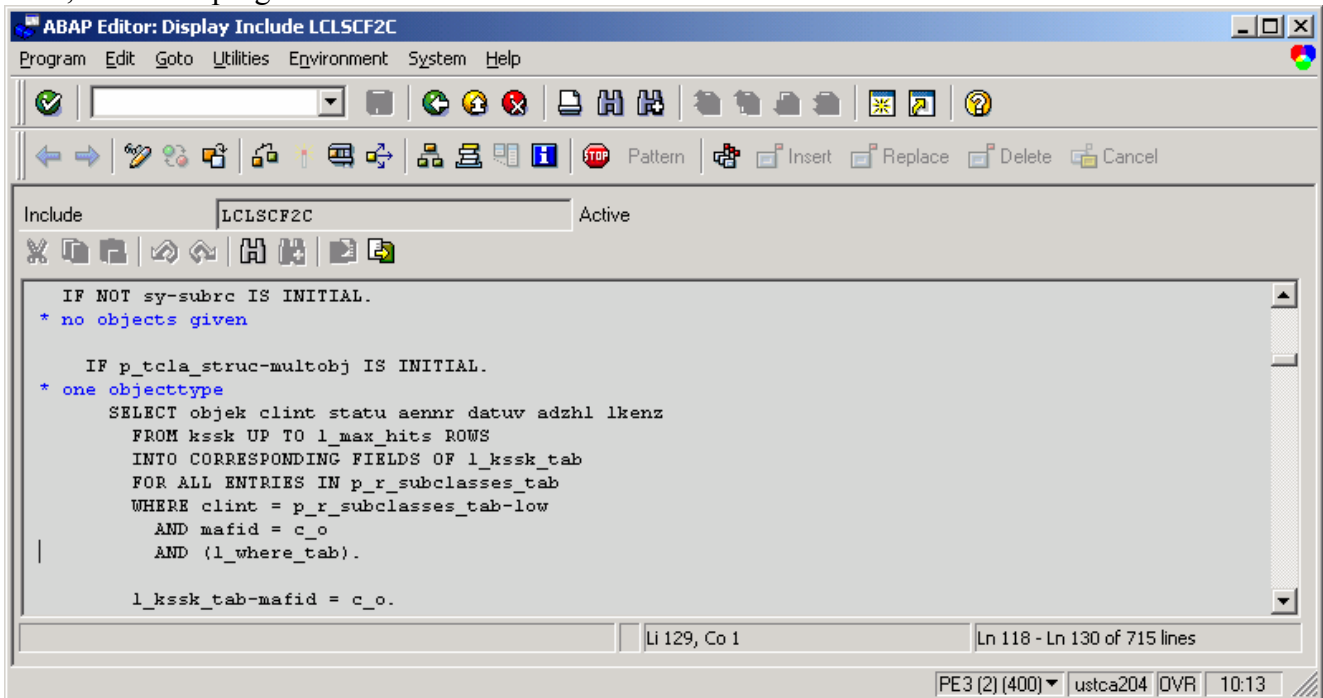
Check ST04 statement cache (will be covered in section 8.3) and see that this statement (the last line in Figure 47) is one of the top statements in total getpages, and that it examined 64M rows and processed (returned) 417 rows. The statement being in the top of the getpage-sorted list shows that this statement is important as a system performance problem, as well as a problem for this transaction.

Check the “per execution” statistics, to see rows examined per execution. (This is not shown in this example). If an SQL statement seldom returns a result row, the ratio rows examined per row processed will make the statement look more inefficient than it is. Rows examined per execution is a better measure of efficiency when rows are seldom returned. Here, since there are 417 tablespace scans, we can see that the statement was executed 417 times, and returned one row each time it was executed.

Synchronous	Synchronous	Getpages	Rows examined	Rows processed	Sorts	Index scans	Tablespace scans	Parallel groups	Statement text
0	0	6913431	4832827	0	0	4496236	0	0	SELECT * FROM "EDUR" WHERE
5680	0	4555229	856792	0	0	0	0	0	DELETE FROM "USERP2" WHERE
110574	0	3889348	13988195	4	0	118	0	0	SELECT T_00 . "VBSLN" FROM
180	0	3073441	101265551	0	0	0	2959	0	SELECT * FROM "VTPA" WHERE
4167	0	2535693	17259953	0	139	0	139	0	SELECT * FROM "V_NTHM" ME
29	0	2455888	29434984	12317	0	12317	0	0	SELECT MAX( "COUNTER" ) FROM
24	0	2034919	3045991	8	0	2899	0	0	SELECT * FROM "MBP" WHERE
204	0	1363662	64119574	417	0	0	417	0	SELECT "OBJEK" , "CLINT" , "

Figure 47: KSSK statement from ST04 cached statement statistics

Next, check the program source from the ST05 trace.

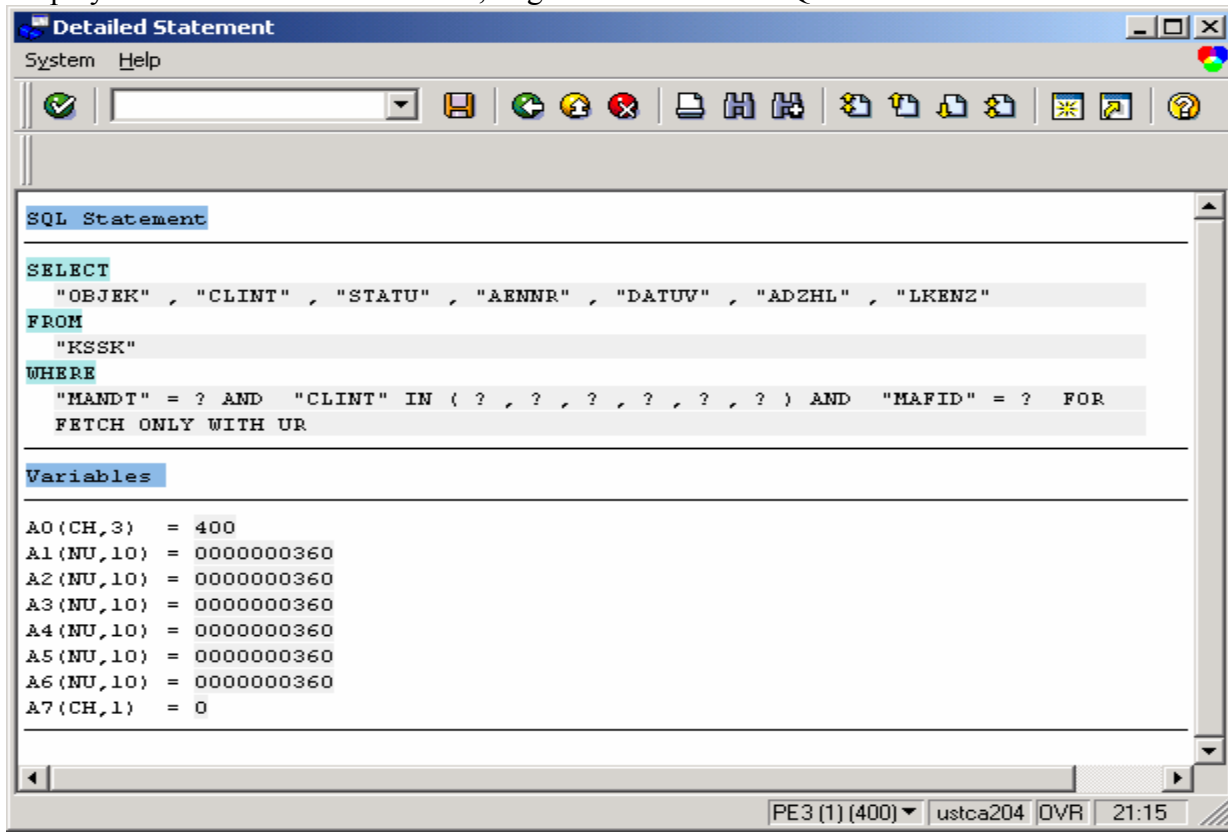


**Figure 48: ST05 source display - ABAP selecting KSSK**

The ABAP is SAP code (LCL...), not user (Y or Z) program.

Note the "SELECT ... FOR ALL ENTRIES". The ABAP "FOR ALL ENTRIES" uses internal tables to specify the predicates. The SAP DBSL (which converts ABAP into database dependent SQL) may convert "FOR ALL ENTRIES" to a DB2 statement with an IN list or a UNION ALL, depending on the selection criteria specified in the "FOR ALL ENTRIES". If there is one column in the predicate, IN list is used. If there is more than one column in the predicate, UNION ALL is used.

Display the statement from ST05 list, to get columns in the SQL where clause so we can check indexes.



```
SQL Statement

SELECT
"OBJEK" , "CLINT" , "STATU" , "AENNR" , "DATUV" , "ADZHL" , "LKENZ"
FROM
"KSSK"
WHERE
"MANDT" = ? AND "CLINT" IN ( ? , ? , ? , ? , ? , ? , ? ) AND "MAFID" = ? FOR
FETCH ONLY WITH UR

Variables

A0(CH,3) = 400
A1(NU,10) = 0000000360
A2(NU,10) = 0000000360
A3(NU,10) = 0000000360
A4(NU,10) = 0000000360
A5(NU,10) = 0000000360
A6(NU,10) = 0000000360
A7(CH,1) = 0
```

Figure 49: ST05 display KSSK statement with variables

The columns are MANDT, CLINT, MAFID.

The ABAP in Figure 48 had “UP TO xxx ROWS”, which might have constrained the number of rows returned from DB2. Check how many rows there are in the table that match the SQL. Use SE16, and put in the values of the variables from Figure 49, then press “number of entries”

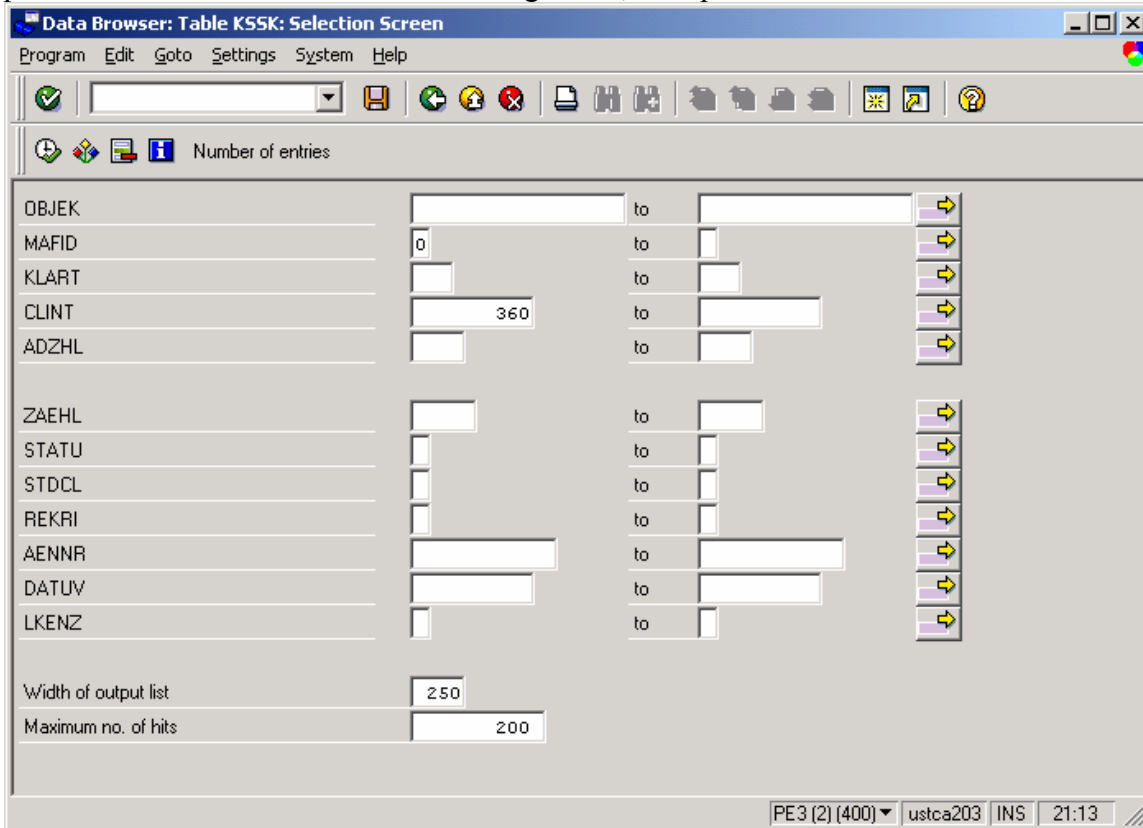
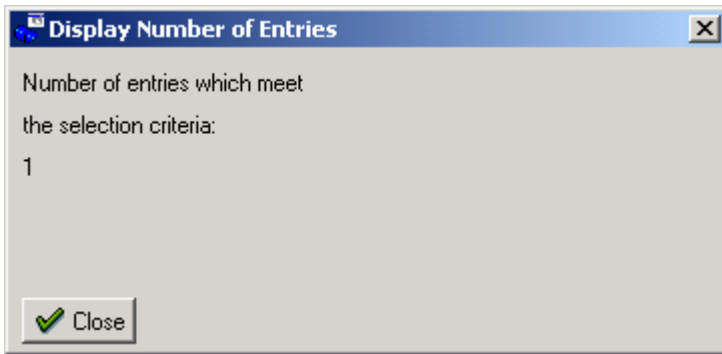
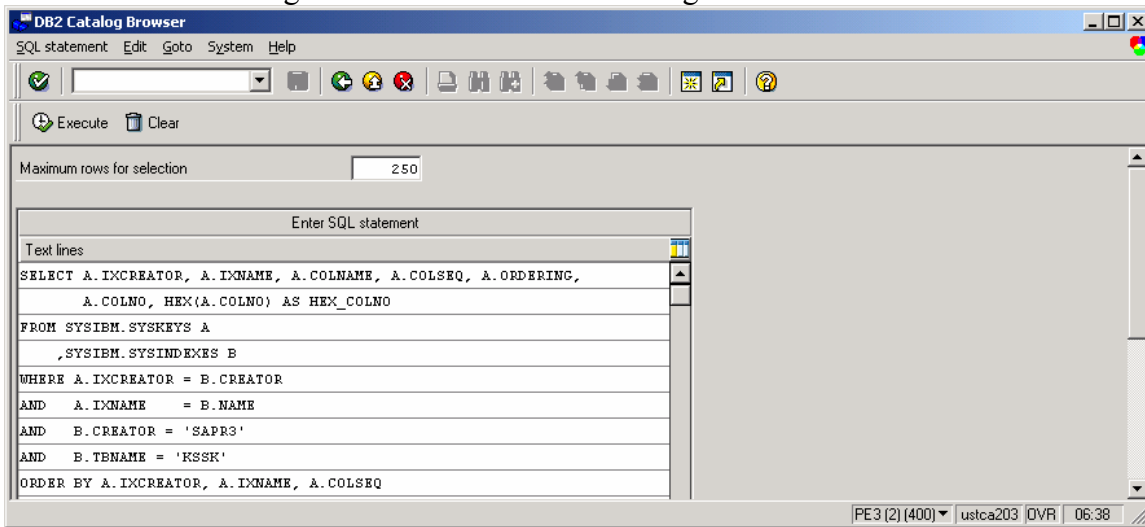


Figure 50: SE16 display table contents



There is only one row that satisfies the predicates with the values specified, so one row is read by this statement.

Use ST04 DB2 catalog browser to check for matching indexes.



**Figure 51: KSSK check indexes and columns**

IXCREATOR	IXNAME	COLNAME	COLSEQ	ORDERING	COLNO	HEX_COLNO
SAPR3	KSSK-N1	MANDT	1	A	1	0001
SAPR3	KSSK-N1	CLINT	2	A	5	0005
SAPR3	KSSK-N1	MAFID	3	A	3	0003
SAPR3	KSSK-0	MANDT	1	A	1	0001
SAPR3	KSSK-0	OBJEK	2	A	2	0002
SAPR3	KSSK-0	MAFID	3	A	3	0003
SAPR3	KSSK-0	KLART	4	A	4	0004
SAPR3	KSSK-0	CLINT	5	A	5	0005
SAPR3	KSSK-0	ADZHL	6	A	6	0006
SAPR3	KSSK-3	MANDT	1	A	1	0001
SAPR3	KSSK-3	AENNR	2	A	11	000B

Figure 52: KSSK indexes and columns

*Index KSSK~N1 is an exact match for the statement.* Why is it not used and why is a tablespace scan done?

Check index cardinality, to see if DB2 intentionally avoided using the index, due to low cardinality.

Maximum rows for selection: 250

```

Enter SQL statement
Text lines
SELECT CREATOR, TBNAME, NAME, COLCOUNT, NLEAF, NLEVELS,
INTEGER(FIRSTKEYCARD) AS FIRSKEYCARD,
INTEGER(FULLKEYCARD) AS FULLKEYCARD
FROM SYSIBM.SYSINDEXES A, SYSIBM.SYSINDEXPART B
WHERE A.CREATOR = 'SAPR3'
AND A.TBNAME = 'KSSK'
AND A.CREATOR = B.IXCREATOR
AND A.NAME = B.IXNAME
    
```

Figure 53: KSSK check index cardinality

CREATOR	TBNAME	NAME	COLCOUNT	NLEAF	NLEVELS	FIRSTKEYCARDF	FULLKEYCARDF
SAPR3	KSSK	KSSK~N1	3	352	3	2	15
SAPR3	KSSK	KSSK~0	6	5,438	3	2	152,577
SAPR3	KSSK	KSSK~3	2	351	3	2	2

Figure 54: KSSK index cardinality

Index KSSK~N1 has cardinality 15, so DB2 thinks that it will not be selective. There are over 150K rows in the table, which can be seen by the cardinality of the KSSK~0 index, since SAP ~0 indexes are unique indexes. Thus, DB2 thinks selects on KSSK~N1 will return many rows. In cases where index access will return many of the rows in a table, a tablespace scan can offer better performance. We can see from the trace and ST04 statement cache statistics that only one row is returned, so we know that it would be better to use the index, in spite of the low index cardinality in the catalog. Here there is a big skew in rows for each value in KSSK~N1. There are only 15 unique values, but one of those 15 unique values returns only one row.

Get the column cardinalities of the predicate columns.

Maximum rows for selection: 250

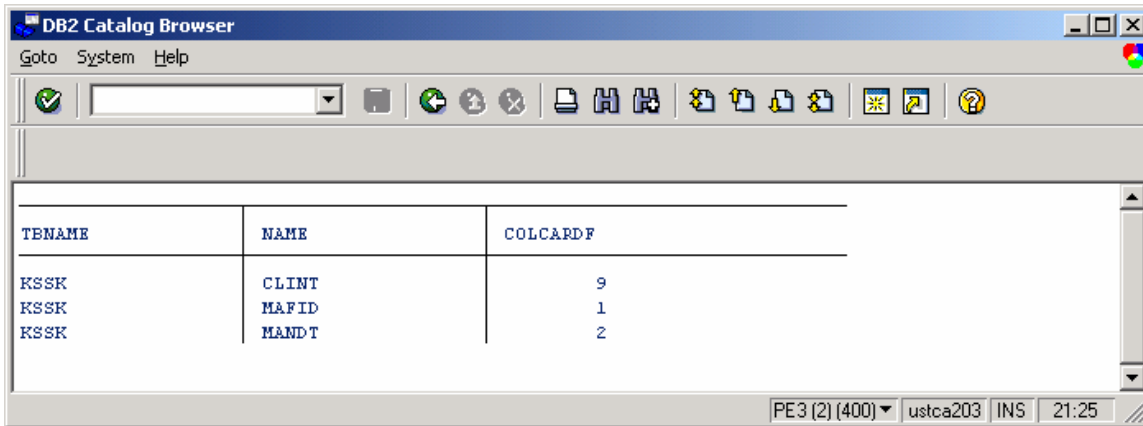
Enter SQL statement

```

Text lines
SELECT TBNAME, NAME, INTEGER(COLCARDF) AS COLCARDF
FROM SYSIBM.SYSCOLUMNS
WHERE TBNAME = 'KSSK' AND
NAME IN ('MANDT', 'CLINT', 'MAFID')
    
```

Figure 55: KSSK check cardinality of columns in KSSK~N1





TENAME	NAME	COLCARDF
KSSK	CLINT	9
KSSK	MAFID	1
KSSK	MANDT	2

No surprise here. If the index N1 has low cardinality, then each of its columns must have low cardinality too.

For sites running SAP 4.5B and later, one can modify the ABAP source to use the hint (USE VALUES FOR OPTIMIZATION) which will cause the statement to be reoptimized at execution, when the variables are passed to DB2. (See SAPnote 162034 for more information on hints with DB2 for OS/390.) With this hint, the DB2 column distribution statistics can be used by the optimizer, and DB2 will know that the program is searching for a value that seldom occurs, and will choose the KSSK~N1 index. RUNSTATS with FREQVAL must be run on KSSK for USE VALUES FOR OPTIMIZATION to be effective.

For sites running SAP versions before 4.5B, one can manually set FULLKEYCARDF on the index KSSK~N1 higher, to make the index more preferred by the optimizer. Since the statement read about 1 million rows for each row returned, we set cardinality to a big number, to make this look like an index that filters well.

```
UPDATE SYSIBM.SYSINDEXES SET FULLKEYCARDF = 100000
WHERE CREATOR = 'SAPR3'
AND TENAME = 'KSSK'
AND NAME = 'KSSK~N1'
```

The disadvantage of setting the statistics is that it can cause the optimizer to choose the N1 index too often, since DB2 will think it is more selective than it really is. In this case, since ST04 showed that each execution of the statement returned one row, this should be a safe change.

*Be careful when manually setting catalog statistics. It can cause DB2 to optimize statements to use the wrong access path. Before and after making the change, use the table filter in ST04 statement cache analysis, to filter and review statements that access the table. Check that no statements are negatively impacted by the change.*

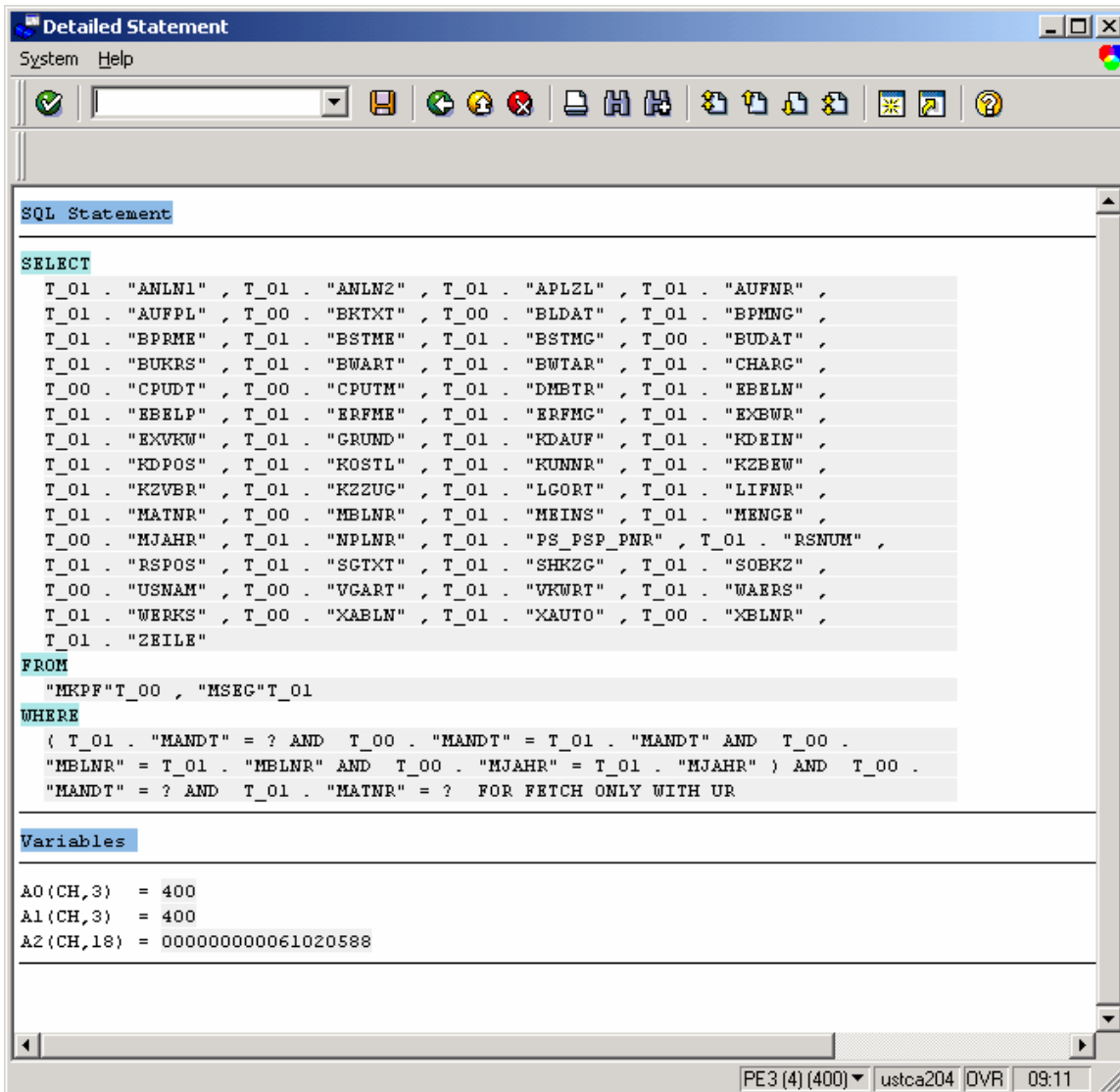
### 7.4.4. Sample transaction analysis - MB51

In this case, we have been asked to examine MB51. Start off with ST05 trace, to examine the SQL. See that there are array fetches against MKPF that take 10-15 ms per row. In general, as mentioned in section 7.1.1.2, one would expect array operations to be much faster – often less than one ms per row.

PID	Typ	Cli	Time	Tcode/prog	Table	SQL op	Recs.	Time
671	DIA	400	15:27:17.477	MB51	MMIM_REP_CUST	SEL	1	1,743
671	DIA	400	15:27:17.502	MB51	- COMMIT WORK -	CW	0	2,214 C
671	DIA	400	15:36:15.187	MB51	MMIM_REP_CUST	SEL	1	17,644
671	DIA	400	15:36:15.238	MB51	- COMMIT WORK -	CW	0	2,561 C
671	DIA	400	15:36:26.785	MB51	MARA	SEL	1	2,264
671	DIA	400	15:36:26.795	MB51	MKPF	SEL	103	1,513,067
671	DIA	400	15:36:28.311	MB51	MAKT	SEL	1	1,551
671	DIA	400	15:36:28.316	MB51	TO01W	SEL	183	56,736
671	DIA	400	15:36:28.375	MB51	USRBF2	SEL	2	14,702
671	DIA	400	15:36:28.506	MB51	LTDX	SEL	1	13,577
671	DIA	400	15:36:28.530	MB51	USRBF2	SEL	0	5,813
671	DIA	400	15:36:28.537	MB51	TRDIR	SEL	1	3,040
671	DIA	400	15:36:28.766	MB51	- COMMIT WORK -	CW	0	4,094 C
671	DIA	400	15:37:08.323	MB51	MMIM_REP_CUST	SEL	1	3,516
671	DIA	400	15:37:08.348	MB51	- COMMIT WORK -	CW	0	2,100 C
671	DIA	400	15:37:14.081	MB51	MARA	SEL	1	3,449
671	DIA	400	15:37:14.090	MB51	MKPF	SEL	80	1,003,989
671	DIA	400	15:37:15.101	MB51	MAKT	SEL	1	1,910
671	DIA	400	15:37:15.104	MB51	TO01W	SEL	183	28,305
671	DIA	400	15:37:15.134	MB51	USRBF2	SEL	2	1,707
671	DIA	400	15:37:15.173	MB51	LTDX	SEL	1	2,752
671	DIA	400	15:37:15.188	MB51	USRBF2	SEL	0	6,366
671	DIA	400	15:37:15.196	MB51	TRDIR	SEL	1	1,654
671	DIA	400	15:37:15.378	MB51	- COMMIT WORK -	CW	0	3,416 C
671	DIA	400	15:43:16.665	MB51	MMIM_REP_CUST	SEL	1	2,605
671	DIA	400	15:43:16.690	MB51	- COMMIT WORK -	CW	0	2,361 C
2421	DIA	400	15:22:43.778	MB51	MMIM_REP_CUST	SEL	1	18,687
2421	DIA	400	15:22:43.840	MB51	- COMMIT WORK -	CW	0	3,792 C
2421	DIA	400	15:26:41.647	MB51	MMIM_REP_CUST	SEL	1	2,748
2421	DIA	400	15:26:41.670	MB51	- COMMIT WORK -	CW	0	2,366 C
2421	DIA	400	15:26:44.955	MB51	MARA	SEL	1	1,844
2421	DIA	400	15:26:44.989	MB51	MKPF	SEL	59	1,224,968
2421	DIA	400	15:26:46.226	MB51	MAKT	SEL	1	4,019
2421	DIA	400	15:26:46.233	MB51	TO01W	SEL	183	84,067
2421	DIA	400	15:26:46.319	MB51	USRBF2	SEL	2	41,712

Figure 56: MB51 - ST05 summarized trace

Display the statement in the ST05 trace list by drilling into the REOPEN, and see that it is a join on MKPF and MSEG. (The ST05 trace just reports one of the tables in a join.) The join is on MBLNR and MJAHR and MANDT. The selection criteria are MATNR and MANDT.



The screenshot shows a window titled "Detailed Statement" with a menu bar (System, Help) and a toolbar. The main area displays an SQL statement with the following structure:

```

SELECT
T_01 . "ANLN1" , T_01 . "ANLN2" , T_01 . "APLZL" , T_01 . "AUFNR" ,
T_01 . "AUFPL" , T_00 . "BKTXT" , T_00 . "BLDAT" , T_01 . "BPMNG" ,
T_01 . "BPRME" , T_01 . "BSTME" , T_01 . "BSTMG" , T_00 . "BUDAT" ,
T_01 . "BUKRS" , T_01 . "BWTAR" , T_01 . "BWTAR" , T_01 . "CHARG" ,
T_00 . "CPUDT" , T_00 . "CPUTM" , T_01 . "DMBTR" , T_01 . "EBELN" ,
T_01 . "EBELP" , T_01 . "ERFME" , T_01 . "ERFMC" , T_01 . "EXBWR" ,
T_01 . "EXVKW" , T_01 . "GRUND" , T_01 . "KDAUF" , T_01 . "KDEIN" ,
T_01 . "KDPOS" , T_01 . "KOSTL" , T_01 . "KUNNR" , T_01 . "KZBEW" ,
T_01 . "KZVBR" , T_01 . "KZZUG" , T_01 . "LGORT" , T_01 . "LIFNR" ,
T_01 . "MATNR" , T_00 . "MBLNR" , T_01 . "MEINS" , T_01 . "MENGE" ,
T_00 . "MJAHR" , T_01 . "NPLNR" , T_01 . "PS_PSP_PNR" , T_01 . "RSNUM" ,
T_01 . "RSPOS" , T_01 . "SGTXT" , T_01 . "SHKZG" , T_01 . "SOBKZ" ,
T_00 . "USNAM" , T_00 . "VGART" , T_01 . "VKWRT" , T_01 . "WAERS" ,
T_01 . "WERKS" , T_00 . "XABLNR" , T_01 . "XAUTO" , T_00 . "XBLNR" ,
T_01 . "ZEILE"
FROM
"MKPF" T_00 , "MSEG" T_01
WHERE
( T_01 . "MANDT" = ? AND T_00 . "MANDT" = T_01 . "MANDT" AND T_00 .
"MBLNR" = T_01 . "MBLNR" AND T_00 . "MJAHR" = T_01 . "MJAHR" ) AND T_00 .
"MANDT" = ? AND T_01 . "MATNR" = ? FOR FETCH ONLY WITH UR

```

Below the SQL statement, the "Variables" section shows:

```

A0(CH,3) = 400
A1(CH,3) = 400
A2(CH,18) = 0000000000061020588

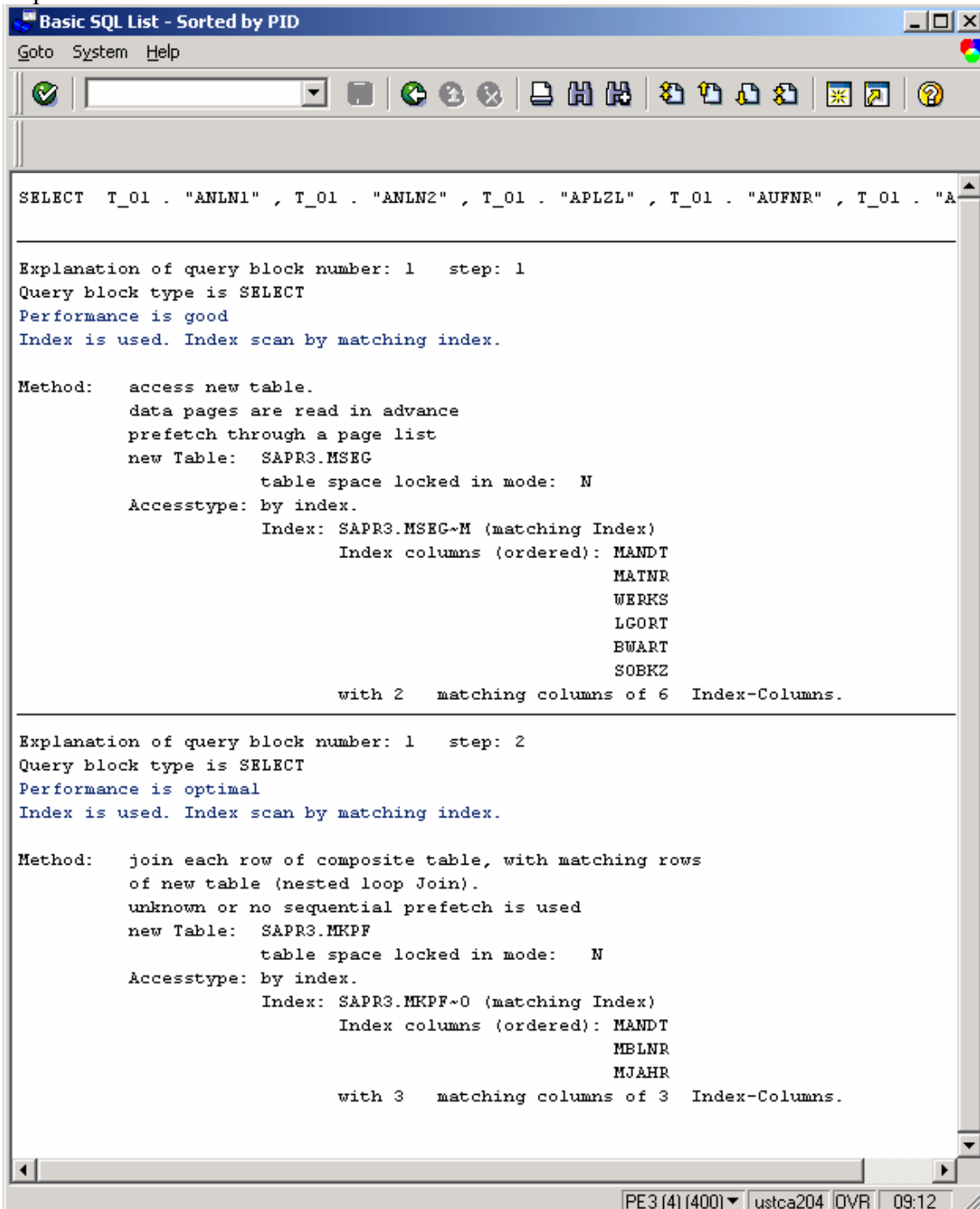
```

The status bar at the bottom of the window displays: PE3 (4) (400) | ustca204 | DWR | 09:11

Note that the join conditions are within parentheses in the WHERE clause, and predicates are outside the parentheses.

If tables are joined in the ABAP, you will see the join conditions in the SQL in ST05. If a database view defined on a join is used by the ABAP, you will not see the join conditions in the SQL. Use SE11 to display the join conditions and other restrictions on a view.

## Explain the statement



```

SELECT T_01 . "ANLN1" , T_01 . "ANLN2" , T_01 . "APLZL" , T_01 . "AUFNR" , T_01 . "A...

Explanation of query block number: 1   step: 1
Query block type is SELECT
Performance is good
Index is used. Index scan by matching index.

Method:   access new table.
          data pages are read in advance
          prefetch through a page list
          new Table:  SAPR3.MSEG
                   table space locked in mode:  N
          Accesstype: by index.
                   Index: SAPR3.MSEG~M (matching Index)
                   Index columns (ordered):  MANDT
                                               MATNR
                                               WERKS
                                               LGORT
                                               BWAET
                                               SOBKZ
                   with 2   matching columns of 6   Index-Columns.

Explanation of query block number: 1   step: 2
Query block type is SELECT
Performance is optimal
Index is used. Index scan by matching index.

Method:   join each row of composite table, with matching rows
          of new table (nested loop Join).
          unknown or no sequential prefetch is used
          new Table:  SAPR3.MKPF
                   table space locked in mode:  N
          Accesstype: by index.
                   Index: SAPR3.MKPF~0 (matching Index)
                   Index columns (ordered):  MANDT
                                               MBLNR
                                               MJAHR
                   with 3   matching columns of 3   Index-Columns.

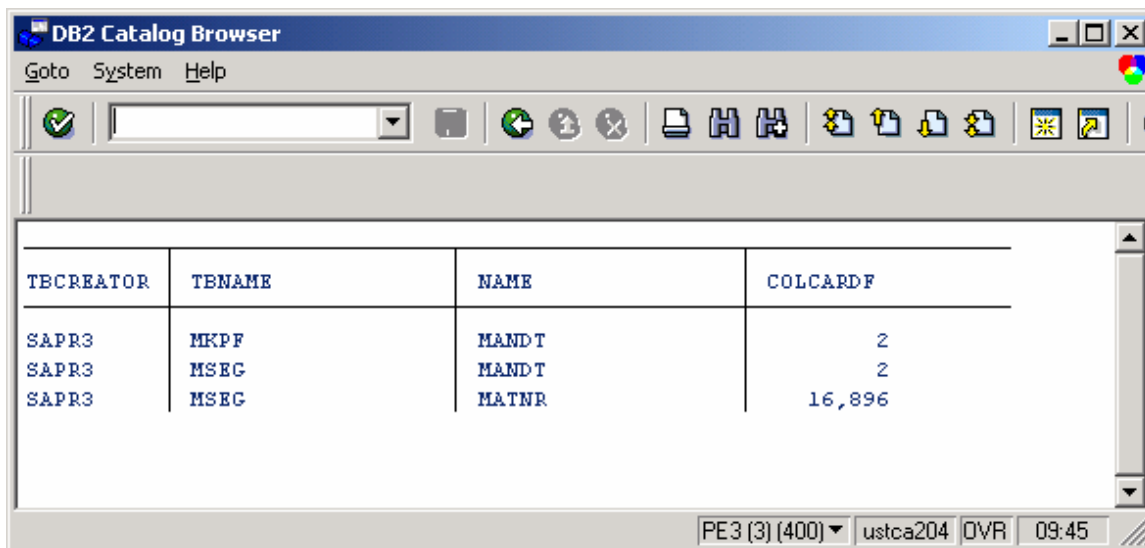
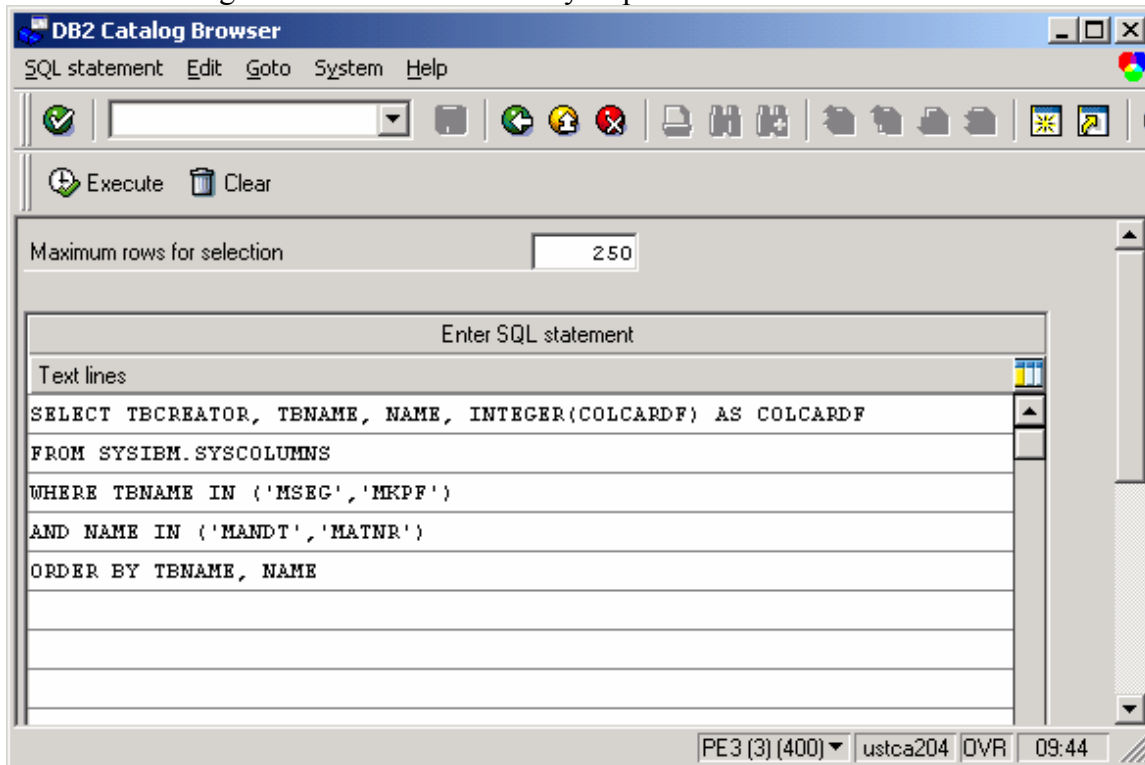
```

PE3 (4) (400) ▼ ustca204 OVR 09:12

Figure 57: MKPF MSEG explain

This looks reasonable. Check the indexes and matching columns. MATNR is a predicate, and is used on the outer table. MBLNR, MJAHR, and MANDT are join conditions, and are all used to access the inner table.

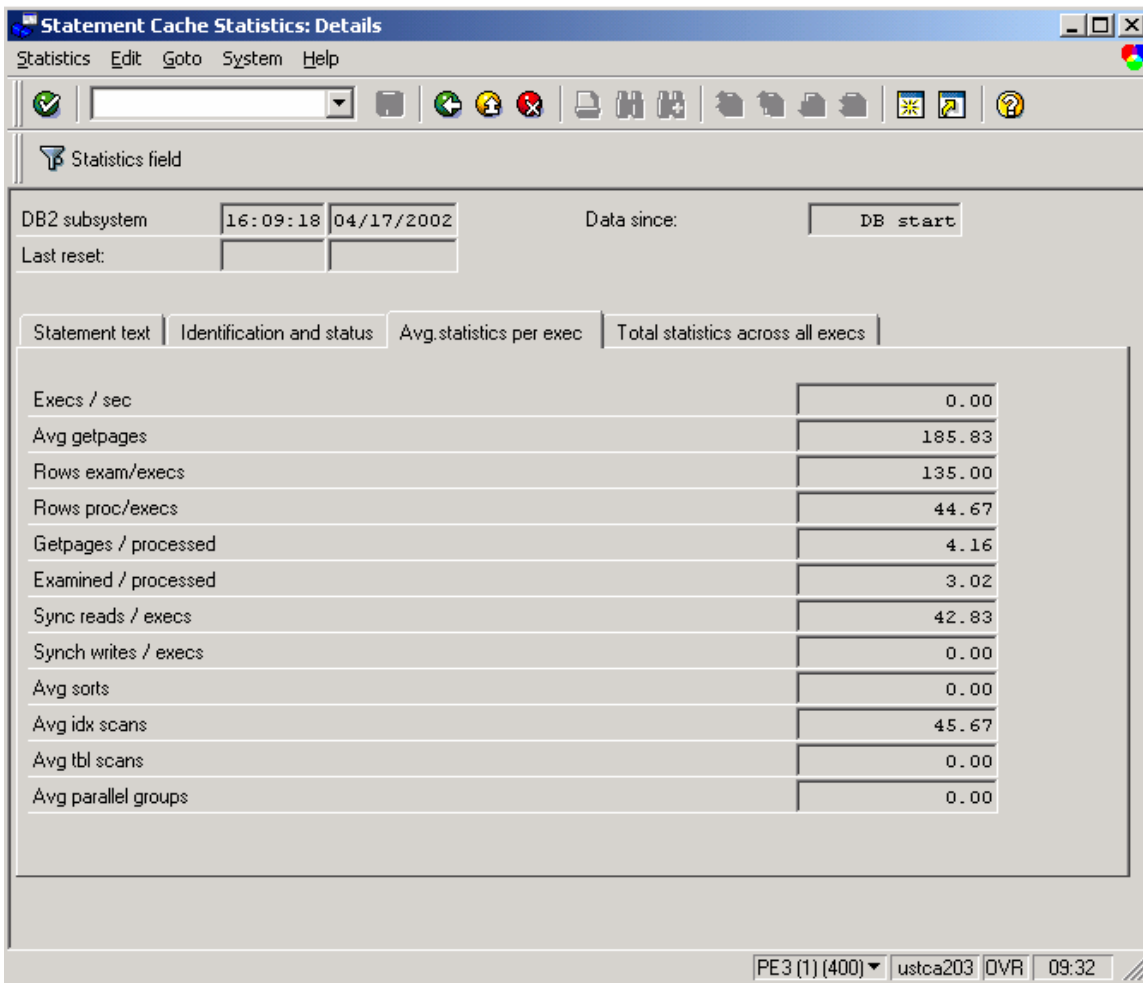
Check the catalog statistics to see cardinality of predicates.



**Figure 58: MB51 predicate cardinality**

MATNR has high cardinality. It was used on outer table. Everything looks fine, why is the statement so slow?

Check the statement statistics.



The screenshot shows a window titled "Statement Cache Statistics: Details" with a menu bar (Statistics, Edit, Goto, System, Help) and a toolbar. Below the toolbar is a "Statistics field" section. The main area displays the following information:

DB2 subsystem: 16:09:18 04/17/2002      Data since: DB start  
 Last reset: [ ] [ ]

Navigation tabs: Statement text | Identification and status | Avg statistics per exec | Total statistics across all execs

Statement text	Avg statistics per exec
Execs / sec	0.00
Avg getpages	185.83
Rows exam/execs	135.00
Rows proc/execs	44.67
Getpages / processed	4.16
Examined / processed	3.02
Sync reads / execs	42.83
Synch writes / execs	0.00
Avg sorts	0.00
Avg idx scans	45.67
Avg tbl scans	0.00
Avg parallel groups	0.00

At the bottom right of the window, it shows: PE3 (1) (400) | ustca203 | DVR | 09:32

185 getpages for 44 rows – about 4 getpages per row – this is OK. Note that there are 42 synchronous reads per execution, almost one per row.

Prefetch I/O is not captured on the statement statistics (since prefetch is not done by the thread, but is done by prefetch processes in DB2). The explain in Figure 57 showed list prefetch being used to access MSEG. Using the synchronous I/O information for the statement, the statistics show that pages needed by this statement are frequently not in memory. (One can calculate a statement hitratio as  $100 * (1 - (\text{synch reads} / \text{getpages}))$  – in this case 77%. Since this does not include prefetch I/O, we know that the pages needed are found in memory less than 77% of the time. If overall bufferpool hitrates in ST04 “subsystem activity” are good, this probably indicates bad re-reference rates in accessing rows in the table.

Check the index stats.

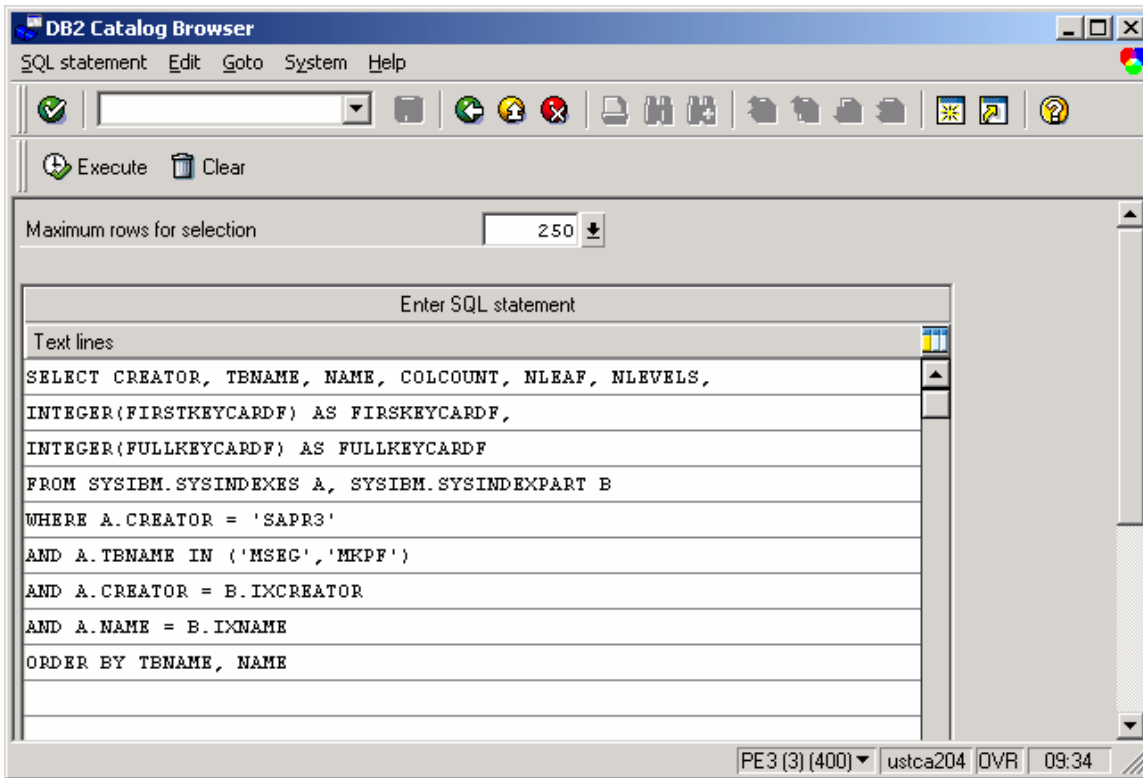


Figure 59: MKPF MSEG query index statistics

The screenshot shows the results of the query in a table format within the DB2 Catalog Browser. The status bar at the bottom shows 'PE3 (3) (400)', 'ustca204 OVR', and '09:35'.

CREATOR	TBNAME	NAME	COLCOUNT	NLEAF	NLEVELS	FIRSKEYCARDF	FULLKEYCARDF
SAPR3	MKPF	MKPF-BUD	3	2,819	3	2	298,355
SAPR3	MKPF	MKPF-ZZ1	2	480	3	2	226
SAPR3	MKPF	MKPF-0	3	2,276	3	2	298,355
SAPR3	MSEG	MSEG-M	6	2,091	3	2	91,765
SAPR3	MSEG	MSEG-R	2	1,092	3	2	2
SAPR3	MSEG	MSEG-S	3	1,102	3	605	1,324
SAPR3	MSEG	MSEG-0	4	5,394	3	2	621,499

Figure 60: MKPF MSEG index statistics

From the FULLKEYCARDF on MSEG (620K) and cardinality of MATNR (16K in Figure 58) we can see that the average material (MATNR) has 38 rows in MSEG (620,000/16,000), about what we saw in the per-execution statement statistics above.

Check size of tables.

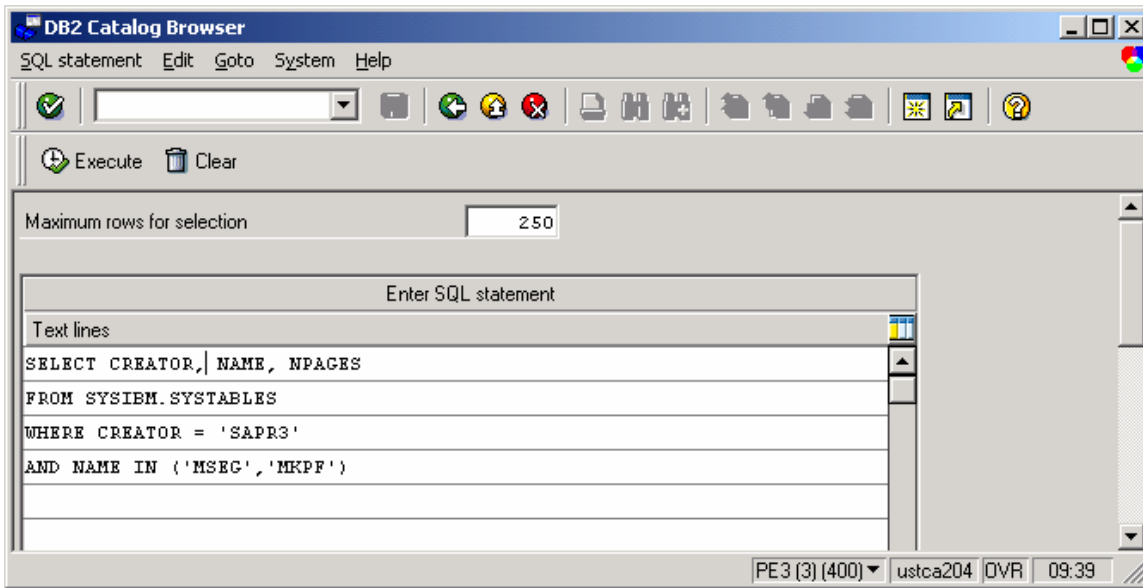


Figure 61: MKPF MSEG query table statistics

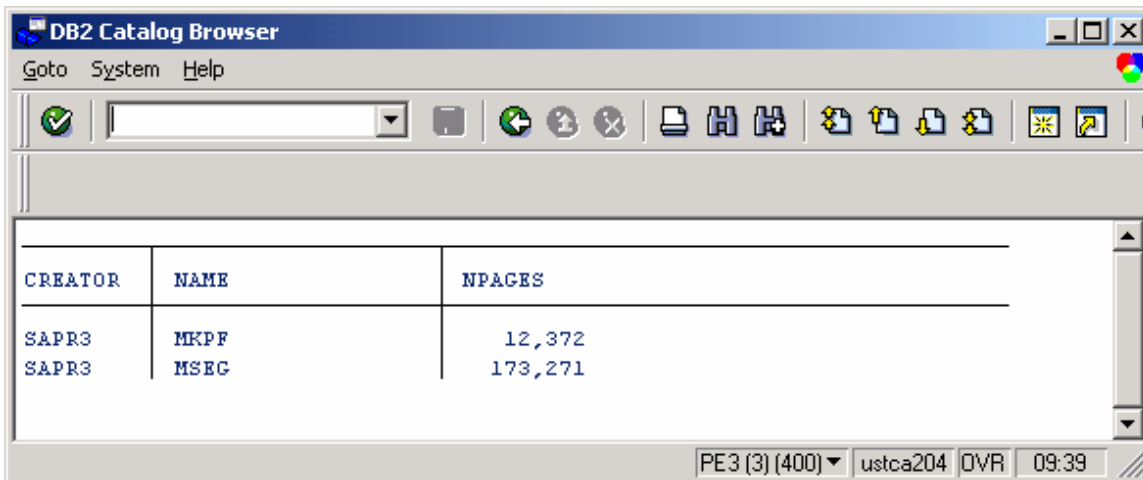


Figure 62: MKPF MSEG table statistics

MSEG is over 700M, which is too large to fit in DB2 buffer memory available.

So, there isn't actually a problem here. From the low hit rate (77%) in the SQL cached statement statistics, it looks like the statement is accessing rows that are widely distributed throughout a large table, and seldom re-referenced. (One would have to do an I/O trace in DB2 to confirm this, but it is pretty clear from the statement statistics.) On average, there is one synchronous I/O operation per row, and that accounts for the long elapsed time of the statement.



*This is an exception to the rule-of-thumb discussed in 7.1.1.2 -- that array operations were often less than one ms per row. (There are always exceptions to ROTs.) Here, the program is doing an array fetch operation using a secondary index to access non-contiguous rows in the table MSEG, which is a very large table. The table is too large to fit in memory, and so there is I/O delay.*

Possible additional steps might be:

- Implement DB2 hardware compression on MSEG and MKPF (if it has not already been done) to get more rows into each block, to enable more of rows to be held in DB2 bufferpools and hiperpools.
- Allocate more memory to bufferpools and hiperpools for this table. (Unless this was an especially critical transaction, it would probably not be worth the memory)
- Examine the I/O performance of the disks where MKPF and MSEG are located, to ensure that there is not contention in the I/O subsystem
- Evaluate changing the clustering sequence on MSEG, so that it is clustered in the sequence of the MSEG~M index. If the most important business access to a table is via a SAP secondary index (MSEG~M in this sample), then one can change the clustering sequence. Access via the new clustering index would then be sequential access, which is generally faster than list prefetch access. *As with changing catalog statistics, evaluate the other programs that access this table, to determine whether the change in clustering sequence will have a negative impact on existing programs.*

## **7.5. Batch**

### **7.5.1. 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. Again, these two SAP parameters should not be enabled all the time, just during detailed problem analysis.

### 7.5.2. Batch analysis starting from STAT records

This problem can be solved with about half the steps shown in the following example, but the screen shots of all the different tools are included to show the different ways that one can approach a problem.

A performance problem with the program REAPRIN0 has been reported. This program (invoice print preparation) is being tested as part of a stress test to prepare for a go-live. In order to achieve the requirements of the batch window, several copies of the program must run in parallel.

As a first step, check STAT records to see what they show.

End time	Tcod	Program	T Scr.	Wp	User	Response time (ms)	Memory used (kB)	Wait time (ms)	CPU time (ms)	DB req. time (ms)
02:10:00		REAPRIN0	B	24	BTCHUSER_ALL	6300370	2,929	0	1658520	4863799
02:10:39		REAPRIN0	B	23	BTCHUSER_ALL	6350161	2,929	0	1712990	4883837
02:11:52		REAPRIN0	B	31	BTCHUSER_ALL	6279285	2,930	0	1800400	4694602

#### Analysis of time in work process

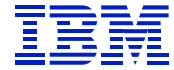
CPU time	1712990 ms	Number	Roll ins	2
RFC+CPIC time	0 ms		Roll outs	0
			Enqueues	3,223
---Response time-----6350161 ms--				
Wait for work process	0 ms	Load time	Program	812 ms
Processing time	1462978 ms		Screen	5 ms
Load time	827 ms		CUA interf.	10 ms
Generating time	0 ms			
Roll (in+wait) time	39 ms	Roll time	Out	0 ms
Database request time	4883837 ms		In	39 ms
Enqueue time	1,240 ms		Wait	0 ms

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

Database requests total	451,476	Request time	4,883,837 ms		
		Matchcode time.	0 ms		
		Commit time	71,963 ms		
Requests on T??? tables	0	Request time	0 ms		
Type of	Database	Requests	Database	Request	Avg.time
ABAP/4 request	Requests	rows	calls	time(ms)	per req.
Total	451476	274,008	90,830	222,241	4883,837
Direct read	146326	36,178	89,157		3028,694
Sequential read	276037	189,454	1,673	173,865	1054,029
Update	12948	30,595		30,595	160,791
Delete	1,611	1,611		1,611	15,994
Insert	14554	16,170		16,170	552,366

Figure 63: REAPRIN0 STAT

Next check the ratio of Database request time to Response time. Database request time is about 75%, so we look at the database request details. It shows that direct read is about 2/3 of total



database request time (3028694 ms of 488387 ms), and that the average direct read is 20 ms. This is very unusual. Since direct read is fully qualified primary index access, direct read times should be very fast - just an ms, or less, for each row. In this case, 89K of 146K requests were satisfied in buffer, so the average direct read time should be very fast.

At this point, there are a number of things that can be done – run an ST05 trace, check ST04 statement cache for long running statements, or just watch the running system and see what is happening. Now run SM66, and look at what the batch processing looks like.

```

-----
| Sort: Server
-----
| Server          No Typ  PID | Status  Reaso Se Start Err   CPU  Time Cli User      Report  Action/Reason for waiting Table
-----
| sbbspaw30_R21_06 28 BTC  18706 | running      Yes          264 010 BTCHUSER_ALL      Direct read      TSP01
| sbbspaw30_R21_06 29 BTC  15812 | running      Yes          264 010 BTCHUSER_ALL      Direct read      TSP01
| sbbspaw30_R21_06 30 BTC  13860 | running      Yes          265 010 BTCHUSER_ALL      Direct read      TSP01
| sbbspaw30_R21_06 31 BTC  14652 | running      Yes          264 010 BTCHUSER_ALL      Direct read      TSP01
| sbbspaw31_R21_06 3 DIA  31502 | running      Yes          1 010 IRUBORMO      SAPLTHFB
| sbbspaw31_R21_06 28 BTC  21846 | running      Yes          265 010 BTCHUSER_ALL SAPLEBUAL Direct read      EKUM
| sbbspaw31_R21_06 29 BTC  24352 | running      Yes          264 010 BTCHUSER_ALL SAPLSTXC
| sbbspaw31_R21_06 30 BTC  19044 | running      Yes          263 010 BTCHUSER_ALL SAPLSPOX Direct read      TSP01
| sbbspaw31_R21_06 31 BTC  24090 | running      Yes          263 010 BTCHUSER_ALL SAPLZCCS Sequential read  EANLH
| sbbspaw32_R21_06 0 DIA  23066 | running      Yes          86 010 OVERMESA      /LBCDWB/ Sequential read  RTTIF
| sbbspaw32_R21_06 28 BTC  18052 | running      Yes          265 010 BTCHUSER_ALL
| sbbspaw32_R21_06 29 BTC  21668 | running      Yes          264 010 BTCHUSER_ALL      Insert          TST03
| sbbspaw32_R21_06 30 BTC  31530 | running      Yes          96 010 BTCHUSER_ALL      Sequential read  BUT100
| sbbspaw32_R21_06 31 BTC  31310 | running      Yes          95 010 BTCHUSER_ALL      Direct read      TSP01
| sbbspaw33_R21_06 0 DIA  39012 | stopped CPIC Yes          1 010 MCCARTUJ      SAPMEMIG CMSEND(SAP) / 43473261
| sbbspaw33_R21_06 10 DIA  22274 | running      Yes          010 WILSORL      SAPMSM66
| sbbspaw33_R21_06 24 BTC  34894 | running      Yes          93 010 BTCHUSER_ALL SAPLE10G Sequential read  V_EGER
| sbbspaw33_R21_06 25 BTC  33538 | running      Yes          90 010 BTCHUSER_ALL SAPLSPOX Direct read      TSP01
| sbbspaw33_R21_06 26 BTC  31690 | running      Yes          90 010 BTCHUSER_ALL SAPLEBUD1 Sequential read  BUT100
| sbbspaw33_R21_06 27 BTC  30416 | running      Yes          90 010 BTCHUSER_ALL SAPLSPOX Direct read      TSP01
| sbbspaw34_R21_06 0 DIA  14986 | running      Yes          1 010 MCCARTUJ
| sbbspaw34_R21_06 20 BTC  29514 | running      Yes          95 010 BTCHUSER_ALL      Sequential read  EPRINT
| sbbspaw34_R21_06 21 BTC  23858 | running      Yes          94 010 BTCHUSER_ALL      Sequential read  V_EGER
| sbbspaw34_R21_06 22 BTC  31180 | running      Yes          94 010 BTCHUSER_ALL      Direct read      USR21
| sbbspaw34_R21_06 23 BTC  32484 | running      Yes          93 010 BTCHUSER_ALL      Direct read      TSP01
-----

```

And note something odd – many of the jobs are shown doing direct read on TSP01.

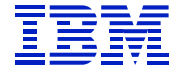
So, run ST05 trace against TSP01.

```

-----
|Transaction =          |PID = 13860 |P type  |Client = 010|User = BTCHUSER_ALL|
-----
|Duration |Object  |DB op. |Rec. |RC |SQL statement
-----
| 4.984|TSP01  |REEXRCS| 1| 0|UPDATE SET "RQCLIENT" = '010', "RQONAME" = 'SCRIPT', "RQ1NAME" = 'RDIP', "
| 48|TSP01  |REOPEN |  | 0|SELECT WHERE "RQIDENT" = 8611 FOR UPDATE OF "RQIDENT" WITH RS KEEP UPDATE LOCKS
| 6.484|TSP01  |FETCH  | 1| 0|
| 5.242|TSP01  |REEXRCS| 1| 0|UPDATE SET "RQCLIENT" = '010', "RQONAME" = 'SCRIPT', "RQ1NAME" = 'RDIP', "
-----
|Transaction =          |PID = 15812 |P type  |Client = 010|User = BTCHUSER_ALL|
-----
|Duration |Object  |DB op. |Rec. |RC |SQL statement
-----
| 574|TSP01  |REOPEN |  | 0|SELECT WHERE "RQCLIENT" = '010' AND "RQONAME" = 'SCRIPT' AND "RQ1NAME" = 'RDIP' AND
| 11.383|TSP01  |FETCH  | 13| 0|
|14691.290|TSP01  |FETCH  | 1| 0|
| 6.534|TSP01  |REEXRCS| 1| 0|UPDATE SET "RQCLIENT" = '010', "RQONAME" = 'SCRIPT', "RQ1NAME" = 'RDIP', "
| 279|TSP01  |REOPEN |  | 0|SELECT WHERE "RQIDENT" = 8611 FOR UPDATE OF "RQIDENT" WITH RS KEEP UPDATE LOCKS
| 8.043|TSP01  |FETCH  | 1| 0|
| 6.784|TSP01  |REEXRCS| 1| 0|UPDATE SET "RQCLIENT" = '010', "RQONAME" = 'SCRIPT', "RQ1NAME" = 'RDIP', "
-----

```

There are many selects against TSP01 that run quickly, but that some are very slow. The slowest statement here took 14 seconds to complete. These are “SELECT FOR UPDATE”, which takes locks, and can be delayed by lock contention. Usually, SAP uses uncommitted read (UR) with DB2 for OS/390, which takes no locks. The ST05 trace can be used to display the ABAP source code, and see the name of the program that is causing the locking problem.



With a locking problem this pervasive, one can often find the table having the locking problem in ST04. Use “ST04 > DB2 subsystem activity > lock waits” to see current lock waiters and holders.

Now check ST04 “global times”, and see that lock/latch delay is very large, over 75% of time in DB2.

CPU in DB2	3.803		
Not attrib. in DB2	9.960		
Suspended in DB2	86.237		
I/O suspension time	6.446	I/O susp (Avg)	6.448
Lock/Latch susp time	75.388	Lock/Latch susp (Avg)	369.75
Other Read Susp	0.810	Othr read susp (avg)	9.437
Othr write susp.	0.071	Othr write susp (avg)	39.562
ServTaskSwitchsusptime	3.270	ServTaskSwitchSusp (Avg)	61.885
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	0.251	Page latch susp (avg)	13.936
Notify mess susp time	0.000	Notify mess susp (Avg)	0
Global lock susp time	0.000	Global lock susp (avg)	0

It looks like we have found the culprit, lock delay on TSP01, but since ST04 “global times” covers the life of the thread, we can run a DB2 suspension trace using IFCID 44,45,226, and 227, then format it with DB2PM “LOCKING REPORT LEVEL(SUSPENSION)”, to check for causes of lock and latch suspensions during a specific periods. Here is an edited report.

SUBSYSTEM: DR21		ORDER: DATABASE	INTERVAL FROM: 05/12/99 16:19:19.51
DB2 VERSION: V5		SCOPE: MEMBER	TO: 05/12/99 17:00:13.24

DATABASE	--- LOCK RESOURCE ---		TOTAL SUSPENDS	--SUSPEND REASONS--			R E S U M E			R E A S O N S		
	TYPE	NAME		LOCAL LATCH	GLOB. IRLMQ	S.NFY OTHER	NORMAL	TIMEOUT/CANCEL	DEADLOCK	AET	AET	AET
ROW	OB =3		12	12	0	0	12	1.155217	0	N/C	0	N/C
	PAGE=X'000013'			0	0	0						
	ROW =X'04'											
ROW	OB =3		11	11	0	0	11	1.344163	0	N/C	0	N/C
	PAGE=X'000013'			0	0	0						
	ROW =X'05'											
ROW	OB =3		148	148	0	0	148	8.286727	0	N/C	0	N/C
	PAGE=X'000013'			0	0	0						
	ROW =X'07'											
ROW	OB =3		179	179	0	0	179	3.658203	0	N/C	0	N/C
	PAGE=X'000013'			0	0	0						
	ROW =X'08'											
ROW	OB =3		99	98	0	0	99	10.676653	0	N/C	0	N/C
	PAGE=X'000013'			1	0	0						
	ROW =X'09'											
TABLE	OB =3		1	0	0	0	1	0.000177	0	N/C	0	N/C
				1	0	0						
** SUM OF TSP01	**		1111	1088	0	0	1111	7.151217	0	N/C	0	N/C
				10	0	13						
*GRAND TOTAL*			4088	1717	0	0	4088	2.046231	0	N/C	0	N/C
				594	0	1777						

Figure 64: DB2PM LOCKING REPORT

There are 4088 total suspensions \* 2.046 sec average time = 8,364 total delay seconds. Then 1111 TSP01 suspensions \* 7.15 sec average time = 7,943 seconds TSP01 delay. TSP01 is causing nearly all the lock/latch delay we saw in ST04.

Direct read was  $\frac{2}{3}$  of database time, and database time was  $\frac{3}{4}$  of elapsed time. Average direct read time was 20ms, where we would expect it to be under 1 ms, since many of the direct reads were buffered. So, we can estimate that we can reduce direct read time by about 95%, and thus decrease runtime by nearly 50% ( $0.95 * \frac{2}{3} * \frac{3}{4}$ ) if we fix the TSP01 problem.

Our action plan, since the program is an SAP program, and the table is a standard SAP table, is to open a note to SAP to report the problem.

### **7.5.3. Sample of SQL cycle analysis in batch**

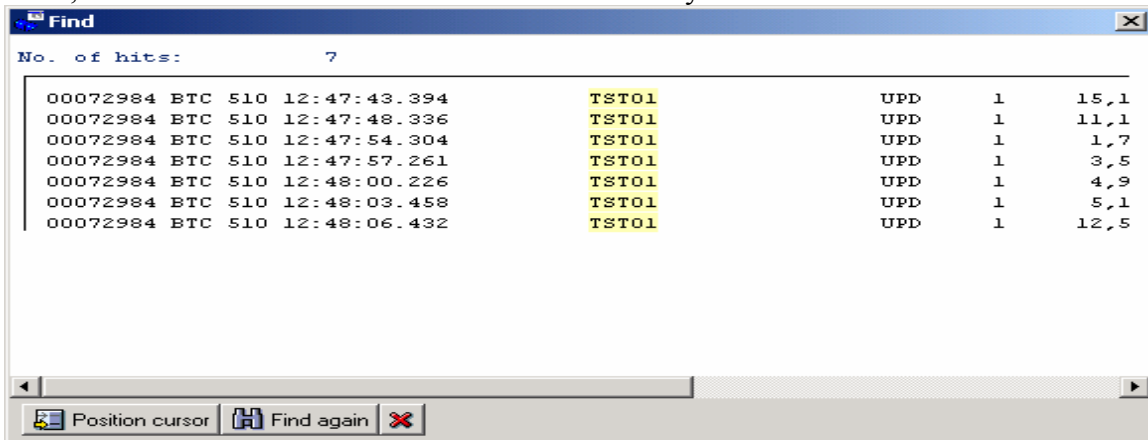
When tracing a batch job, it is important to find the cyclical behavior of the job, so that analysis of the SQL can be performed through at least a full cycle. Analyzing and aggregating several cycles is preferable, in order to average out the impact of transient conditions.

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”.

PID	Pro	Cli	Time	Tcode/pr	Table	SQL op	Recs.	Time
00072984	BTC	510	12:47:43.251		ARFCSDATA	INS	5	23,685
00072984	BTC	510	12:47:43.275		- COMMIT WORK -	CW	0	25,741 C
00072984	BTC	510	12:47:43.394		TST01	UPD	1	15,192
00072984	BTC	510	12:47:43.410		- COMMIT WORK -	CW	0	9,397 C
00072984	BTC	510	12:47:43.421		NAST	SEL	0	83
00072984	BTC	510	12:47:44.806		VBUK	SEL	1	43,513
00072984	BTC	510	12:47:44.866		VBPA	SEL	1	60,675
00072984	BTC	510	12:47:44.928		KNAL	SEL	1	34,985
00072984	BTC	510	12:47:44.964		VBK	SEL	1	2,036
00072984	BTC	510	12:47:44.969		KNVV	SEL	1	3,848
00072984	BTC	510	12:47:44.974		EIKP	SEL	1	31,555
00072984	BTC	510	12:47:45.006		VBPP	SEL	8	114,630
00072984	BTC	510	12:47:45.123		EIPO	SEL	8	84,872
00072984	BTC	510	12:47:45.227		VBAP	SEL	1	21,064
00072984	BTC	510	12:47:45.249		VBAP	SEL	1	26,377
00072984	BTC	510	12:47:45.276		VBAP	SEL	1	3,756
00072984	BTC	510	12:47:45.281		VBAP	SEL	1	2,461
00072984	BTC	510	12:47:45.284		VBAP	SEL	1	2,172
00072984	BTC	510	12:47:45.288		VBAP	SEL	1	10,324
00072984	BTC	510	12:47:45.299		VBAP	SEL	1	1,953
00072984	BTC	510	12:47:45.302		VBAP	SEL	1	2,507
00072984	BTC	510	12:47:45.321		VBPA	SEL	1	1,988
00072984	BTC	510	12:47:45.325		VBAP	SEL	12	46,937
00072984	BTC	510	12:47:45.374		VBAK	SEL	1	2,636
00072984	BTC	510	12:47:45.378		LIKP	SEL	1	6,637
00072984	BTC	510	12:47:45.391		KNAL	SEL	1	2,963
00072984	BTC	510	12:47:45.395		KNVV	SEL	3	2,059
00072984	BTC	510	12:47:45.398		KNB1	SEL	1	24,284
00072984	BTC	510	12:47:45.423		SADR	SEL	1	2,116
00072984	BTC	510	12:47:45.426		VBPA	SEL	1	1,645
00072984	BTC	510	12:47:45.429		STXH	SEL	0	63,392
00072984	BTC	510	12:47:45.493		STXH	SEL	0	12,635
00072984	BTC	510	12:47:45.507		STXH	SEL	0	11,336
00072984	BTC	510	12:47:45.519		STXH	SEL	0	1,683
00072984	BTC	510	12:47:45.521		LIKP	SEL	1	12,132
00072984	BTC	510	12:47:45.534		VBK	SEL	1	3,813
00072984	BTC	510	12:47:45.539		VBFA	SEL	1	88,814
00072984	BTC	510	12:47:45.628		VTTK	SEL	1	1,768

Figure 65: TST01 - Select starting point in summarized ST05 trace

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



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

PID	Pro Cli Time	Tcode/pr Table	SQL op Recs.	Time
00072984	BTC 510 12:48:03.269	ARFCSDATA	INS 4	25,930
00072984	BTC 510 12:48:03.295	- COMMIT WORK -	CW 0	71,520 C
00072984	BTC 510 12:48:03.458	TST01	UPD 1	5,131
00072984	BTC 510 12:48:03.464	- COMMIT WORK -	CW 0	15,328 C
00072984	BTC 510 12:48:03.480	NAST	SEL 0	87
00072984	BTC 510 12:48:04.858	VBUK	SEL 1	2,074
00072984	BTC 510 12:48:04.877	VBPA	SEL 1	1,874
00072984	BTC 510 12:48:04.880	VBK	SEL 1	4,966
00072984	BTC 510 12:48:04.889	VBK	SEL 3	30,317
00072984	BTC 510 12:48:04.940	VBAP	SEL 1	31,259
00072984	BTC 510 12:48:04.972	VBAP	SEL 1	4,332
00072984	BTC 510 12:48:04.977	VBAP	SEL 1	2,956
00072984	BTC 510 12:48:04.997	VBPA	SEL 1	3,702
00072984	BTC 510 12:48:05.002	SADR	SEL 1	39,882
00072984	BTC 510 12:48:05.043	VBAP	SEL 3	4,203
00072984	BTC 510 12:48:05.049	VBK	SEL 1	2,294
00072984	BTC 510 12:48:05.052	LIK	SEL 1	8,771
00072984	BTC 510 12:48:05.064	KNA1	SEL 1	2,934
00072984	BTC 510 12:48:05.068	KNVV	SEL 3	1,974
00072984	BTC 510 12:48:05.071	KNE1	SEL 1	1,397
00072984	BTC 510 12:48:05.073	SADR	SEL 1	1,263
00072984	BTC 510 12:48:05.075	VBPA	SEL 1	1,378
00072984	BTC 510 12:48:05.077	STXH	SEL 0	2,345
00072984	BTC 510 12:48:05.080	STXH	SEL 1	1,644
00072984	BTC 510 12:48:05.083	STXL	SEL 1	4,584
00072984	BTC 510 12:48:05.089	STXH	SEL 0	1,634
00072984	BTC 510 12:48:05.091	LIK	SEL 1	25,082
00072984	BTC 510 12:48:05.117	VBK	SEL 1	1,944
00072984	BTC 510 12:48:05.119	VBFA	SEL 1	15,324
00072984	BTC 510 12:48:05.135	VTTK	SEL 1	30,807
00072984	BTC 510 12:48:05.166	LFAL	SEL 1	1,614
00072984	BTC 510 12:48:05.169	STXH	SEL 0	40,454
00072984	BTC 510 12:48:05.210	KNA1	SEL 1	1,350
00072984	BTC 510 12:48:05.212	VBK	SEL 1	2,430
00072984	BTC 510 12:48:05.215	KNKK	SEL 1	2,506
00072984	BTC 510 12:48:05.218	BSID	SEL 1	4,943
00072984	BTC 510 12:48:05.225	STXH	SEL 0	1,810
00072984	BTC 510 12:48:05.227	KNVV	SEL 1	1,822

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.

Set new transaction code

Tcode/program: 5loop

Continue Cancel



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-TCPO2	SEL	10	1,690	280,318	4.4
SLOOP	VBFA	SEL	6	5	252,698	3.9
SLOOP	MAST	SEL	10	6	230,069	3.6
SLOOP	KOCLU	SEL	5	5	212,158	3.3
SLOOP	MAST	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-TCPO0	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	KMB1	SEL	5	5	57,362	0.9
SLOOP	EIKP	SEL	2	2	54,976	0.9
SLOOP	KMVV	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	LF01	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	VERK	SEL	13	12	35,422	0.6
SLOOP	KMKK	SEL	10	10	34,406	0.5
<b>Total</b>			<b>491</b>	<b>2,388</b>	<b>6,436,007</b>	<b>100.0</b>

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 7.4.2, there is not a huge problem on any of the tables, and one can only gain a small incremental improvement by addressing the slow SQL against the VBAP and VBRP tables.

As in the transaction examples in section 7.4, use ST05 to explain long running SQL statements, use ST04 DB2 catalog browser to check whether a better index is available, or if a new index might be needed.

### 7.5.4. Sample end-to-end batch time analysis

In cases where the STAT records do not contain valid data, and we need to fully characterize where the batch jobs spend most of their processing time, we can evaluate the end-to-end components of batch response time by combining ST04 thread statistics with ST06 CPU time.

In this case, we will analyze the background job running in work process 0, PID 99028. We need the work process number for the ST04 thread display, and the PID for ST06. Both can be gotten with SM50.

The screenshot shows the 'Process Overview' window with a table of process details. The table has the following columns: No., Ty., PID, Status, ReasonStart, Err Sem, CPU, Time, Program, ClieUser, and Action. The data rows are as follows:

No.	Ty.	PID	Status	ReasonStart	Err Sem	CPU	Time	Program	ClieUser	Action
0	BTC	99028	running	Yes	1	66:47	1604	SAPLZATP	510 EDIBATCH	
1	BTC	78368	waiting	Yes		52:15				
2	BTC	87396	waiting	Yes		6:09				
3	DIA	85272	waiting	Yes	1	6:51				
4	BTC	113950	waiting	Yes		6:41				
5	DIA	41896	waiting	Yes		5:07				
6	DIA	121074	waiting	Yes		4:44				
7	DIA	106580	waiting	Yes		2:52				
8	DIA	54502	running	Yes		2:14		RSMON000	510 GORDOMR	
9	UPD	72074	waiting	Yes		4:05				
10	BTC	124638	waiting	Yes		8:01				
11	BTC	98728	waiting	Yes		5:36				
12	BTC	26820	running	Yes		1:26	710	RGUDELPC	510 MOAYC	Delete
13	DIA	105866	waiting	Yes		1:43				
14	UPD	112194	waiting	Yes		0:19				
15	UPD	73660	waiting	Yes		0:02				
16	DIA	61116	waiting	Yes		0:07				
17	DIA	92360	waiting	Yes		0:01				
18	SPO	31590	waiting	Yes		0:01				
19	UP2	81020	waiting	Yes		4:55				

Figure 66: SM50 display

Get starting point data for the PID: “ST06 > detail analysis > top CPU”. Note the CPU time (54 minutes and 10 seconds) or print/save the screen for later.

Pid	Username	Command	CPU Util (%)	CPU Time (s)	Resident size (kB)	Prior.
80,406	root	disptwork	378.57	52:	21,328	64
79,450	root	lastat	9.20	1:1	92	64
95,086		/usr/sap/NPD/SYS/ex	1.97	14105:3	2,060	60
99,028	npdadm	dw.sapNPD_D00 pf=/u	0.79	54:10	22,732	60
121074	npdadm	dw.sapNPD_D00 pf=/u	0.24	4:28	12,604	60
106580	npdadm	dw.sapNPD_D00 pf=/u	0.18	1:47	12,492	60
85,272	npdadm	dw.sapNPD_D00 pf=/u	0.18	6:16	13,184	69
3,402	root	/usr/sbin/syned 60	0.04	2787:20	292	60

Get the starting point for the thread: “ST04 > thread activity > choose the thread > times”. Save/print the screen for later.

Subsystem: NPD2    At: 14:41:44    03/26/2002

Job name: NPDPIP53    Correlation ID: NPDPIP53

DB2 Plan: H0503000    Connection ID: DB2CALL

Auth ID: ICLI    Status:

Times			
Activity time	4:01:18.925425		
Time out of DB2	3:02:19.328414	75.553	% of activity time
Total CPU time	19:39.432430	8.146	% of activity time
Time spent in DB2	58:59.597011	24.447	% of activity time
CPU time in DB2	19:05.079990	32.351	% of time in DB2
Suspended in DB2	28:33.513433	48.410	% of time in DB2
Not attrib. in DB2	11:21.003588	19.240	% of time in DB2

Suspension times and events			
	Time [hh:mm:ss] or [ddd hh:mm]	Events	Avg duration [ ms ]
I/O suspension	17:45.434843	68887	15.466
Lock/Latch susp	14.505274	5441	2.666
Other Read susp	9:47.752411	20326	28.916
Other Write susp	0.036279	2	18.140

Then gather ST04 and ST06 ending points using the same process as above. Unlike the ST05 cycle process in section 7.5.3, this method does not correlate the monitoring to a cycle in the job, so just let it run a few minutes.

Pid	Username	Command	CPU Util [%]	CPU Time [s]	Resident size [kB]	Prior.
128298	root	gvrd	391.12	74:5	2,400	64
99,028	npdadm	dw.sapNPD_D00 pf=/u	2.14	76:13	22,736	101
96,086		/usr/sap/NPD/SYS/ex	1.35	14120:4	2,060	60
81,020	npdadm	dw.sapNPD_D00 pf=/u	0.06	5: 3	15,104	60
72,074	npdadm	dw.sapNPD_D00 pf=/u	0.03	4:11	19,460	60
120564	root	qstatus	0.01	0:	268	64
19,612	root	/usr/sbin/rsct/bin/	0.01	927:16	2,076	38
118270	root	reabak	0.00	0:	440	64
97,946	npdadm	R3trans -d	0.00	0: 0	1,508	60

Subsystem: **NPD2** At: **15:08:15** on **03/26/2002**

Job name: **NPDPIP53** Correlation ID: **NPDPIP53**

DB2 Plan: **H0503000** Connection ID: **DB2CALL**

Auth ID: **ICLI** Status:

### Times

Activity time	4:27:49.396765		
Time out of DB2	3:26:28.119720	77.091	% of activity time
Total CPU time	19:54.364673	7.433	% of activity time
Time spent in DB2	1:01:21.277045	22.909	% of activity time
CPU time in DB2	19:17.548921	31.444	% of time in DB2
Suspended in DB2	30:33.273138	49.800	% of time in DB2
Not attrib. in DB2	11:30.454986	18.756	% of time in DB2

### Suspension times and events

	Time [hh:mm:ss] or [ddd hh:mm]	Events	Avg duration [ ms ]
I/O suspension	19:03.653494	74685	15.313
Lock/Latch susp	15.767021	5800	2.718
Other Read susp	10:28.026479	21951	28.610
Other Write susp	0.036279	2	18.140

Now, get out the time calculator, and fill in the table with the information from ST04 and ST06:

	Start	End	Delta
Activity Time (elapsed)	4:01:18.92	4:27:49.39	26:30.07
CPU on app server (ST06)	54:10.00	76:13.00	22:13.00
Total CPU time (ST04)	19:39.43	19:54.36	00:14.07
Suspended in DB2 (ST04)	28:33.51	30:33.27	00:59.36
Not attrib in DB2 (ST04)	11:21.00	11:30.45	00:09.45
Overall not accounted (calculated)			02:53.12

In this example, while the DB2 delay is large relative to CPU in DB2 (59 to 14), CPU on the application server is the largest amount of time, and would be the first step in improving performance. SE30 should be used for further analysis of the program.

The “overall not accounted” time (which we calculate by subtracting CPU, suspend, and “not attrib.” from elapsed) would include any of the “missing time” elements from STAT (section 7.3.12), as well as network time between application server and database server.

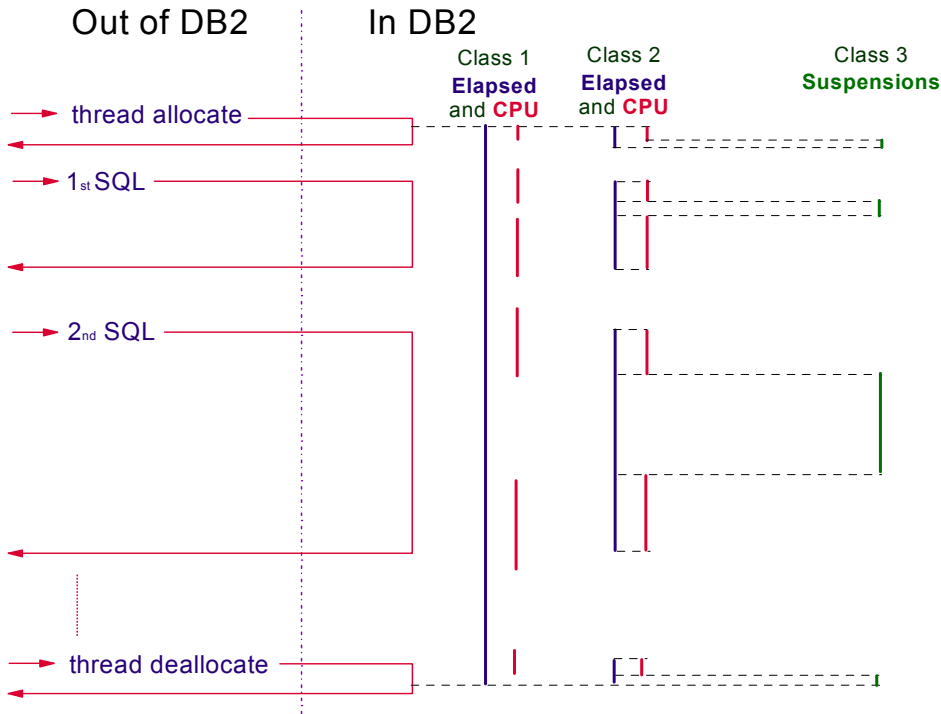
If suspend time were large, look at the individual suspend categories (I/O, lock, etc.) to find the source, and use ST05, as in section 7.5.2, to look for inefficient statements.

## 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 67: DB2 time categories**

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 “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 there 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 subsystem 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.



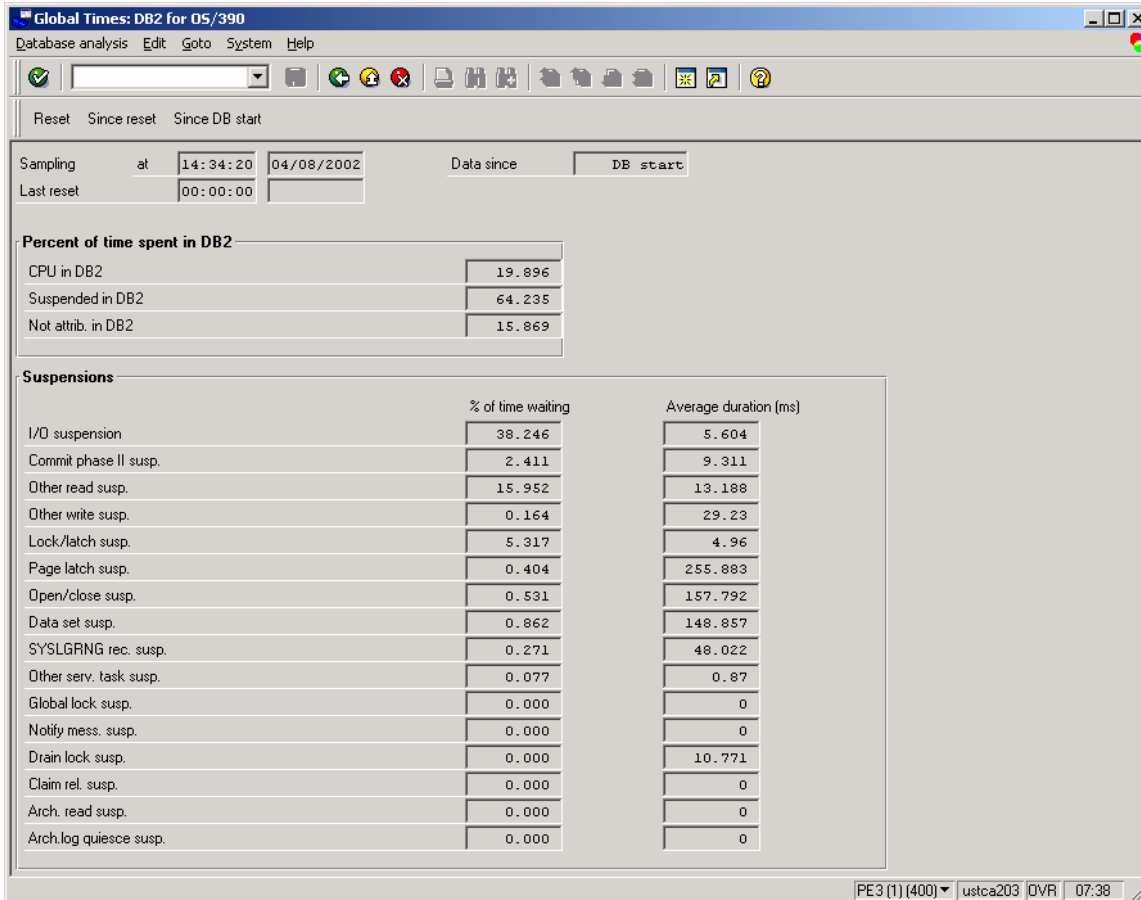


Figure 68: ST04 global times

### 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 done by the thread running the 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 tablespaces 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 8.3. 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 OS/390 tools such as RMF III and RMF 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 workloads
  - 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. This can be done using 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 followed by SE11 “where used” or by reviewing STAT data.
  - 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 times

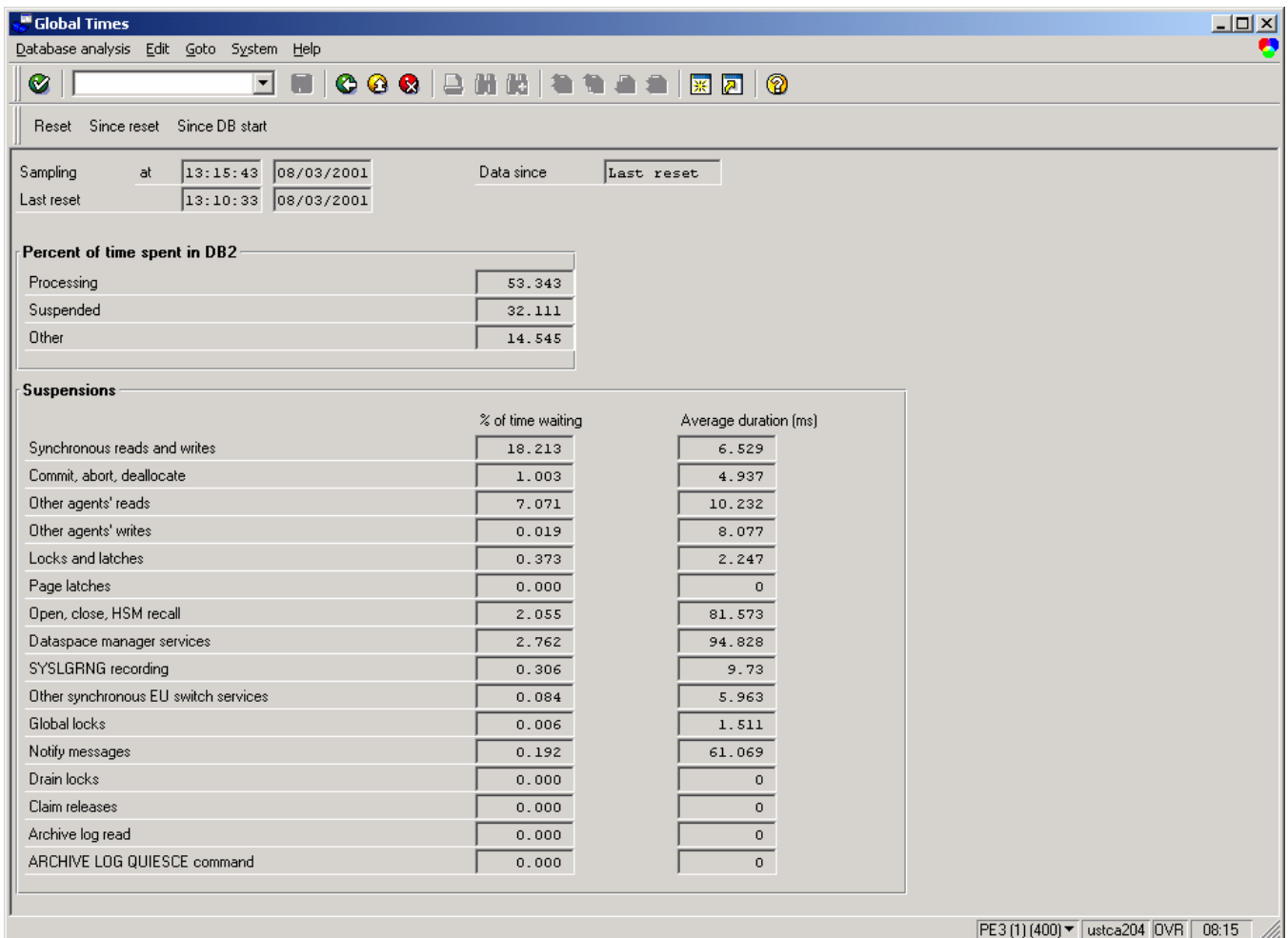


Figure 69: Good ST04 times

In Figure 69, 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 OS/390 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

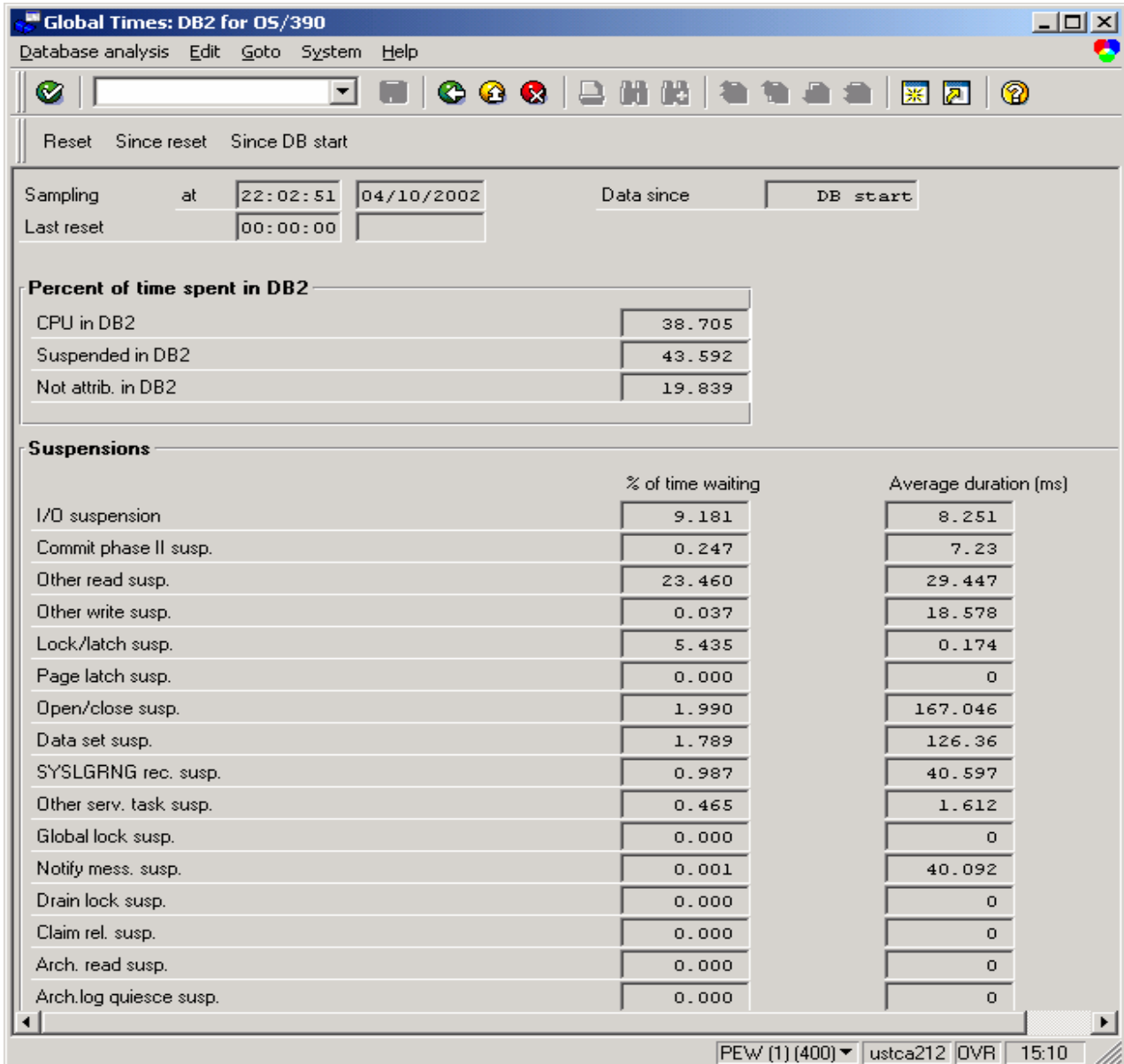


Figure 70: ST04 with long total “other read suspension”

In Figure 70, the delay time is about 61% (43.5+19.8) with CPU about 39%, which is within the normal range. But there is much more “other read” (prefetch) suspension than there is I/O (synchronous) suspension. This is a bit unusual. “Other read” time is frequently only 1/3 to 2/3 of I/O suspension (synchronous I/O). Here, there may be a situation where inefficient SQL is scanning lots of pages via prefetch, and is causing higher CPU utilization (which makes the delay ratio look better). We need to check the SQL cache, and determine why there is so much prefetch activity.

This suspension time profile may be normal for a system that is largely used for reporting, rather than transaction processing. SAP reporting SQL generates access paths using prefetch more often than SAP transaction SQL does. Transactions generally retrieve just a few rows per database call. Reports can retrieve hundreds or thousands of rows per DB call. DB2 may optimize these these statements to use sequential prefetch, to optimize access to large amounts of data.

### 8.2.3. ST04 Times points to constraint on DB server

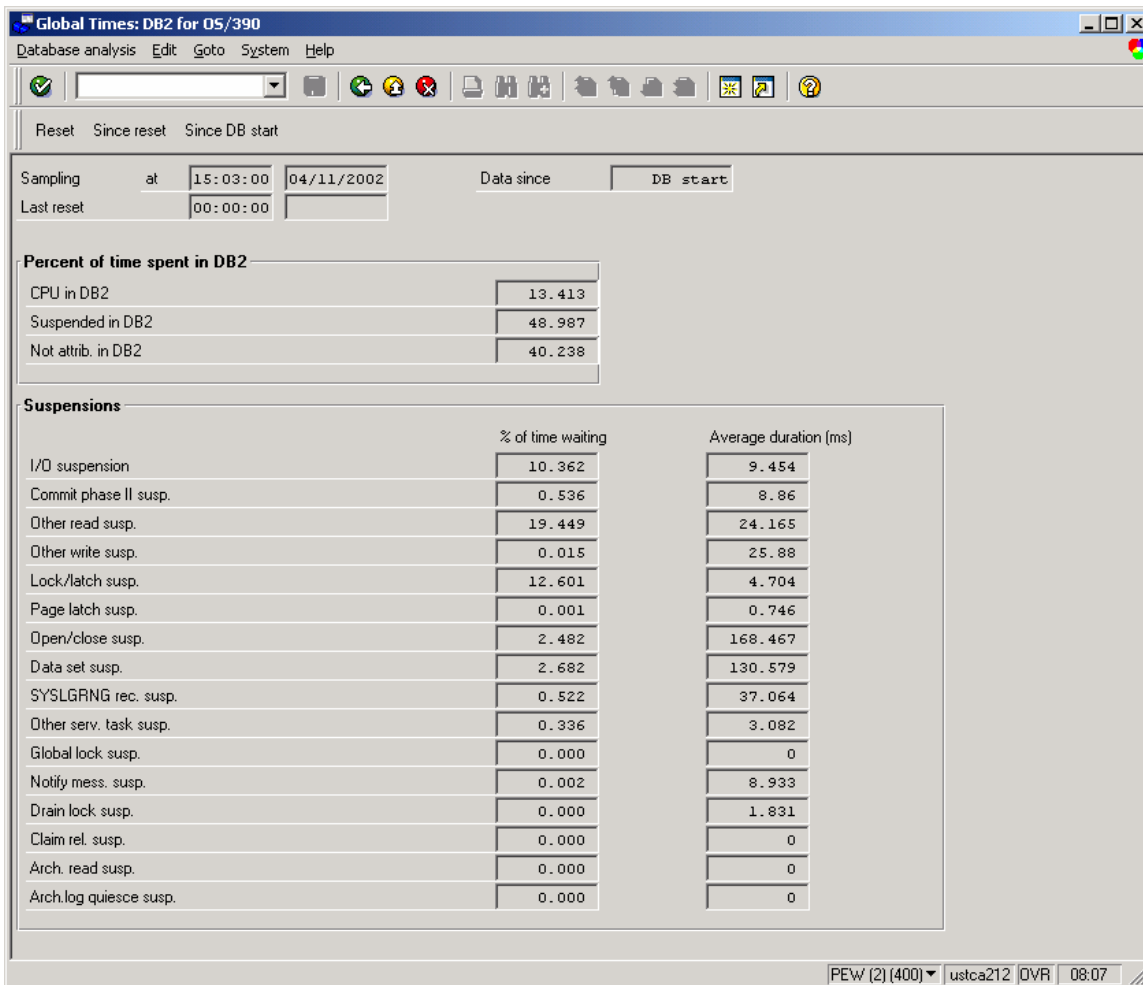


Figure 71: ST04 times with high “Not attrib. in DB2”

ST04 times shows “Not attributed” time is very high – 40%. This points to a problem on the database server -- usually a storage or CPU constraint. From SAP, we can 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)

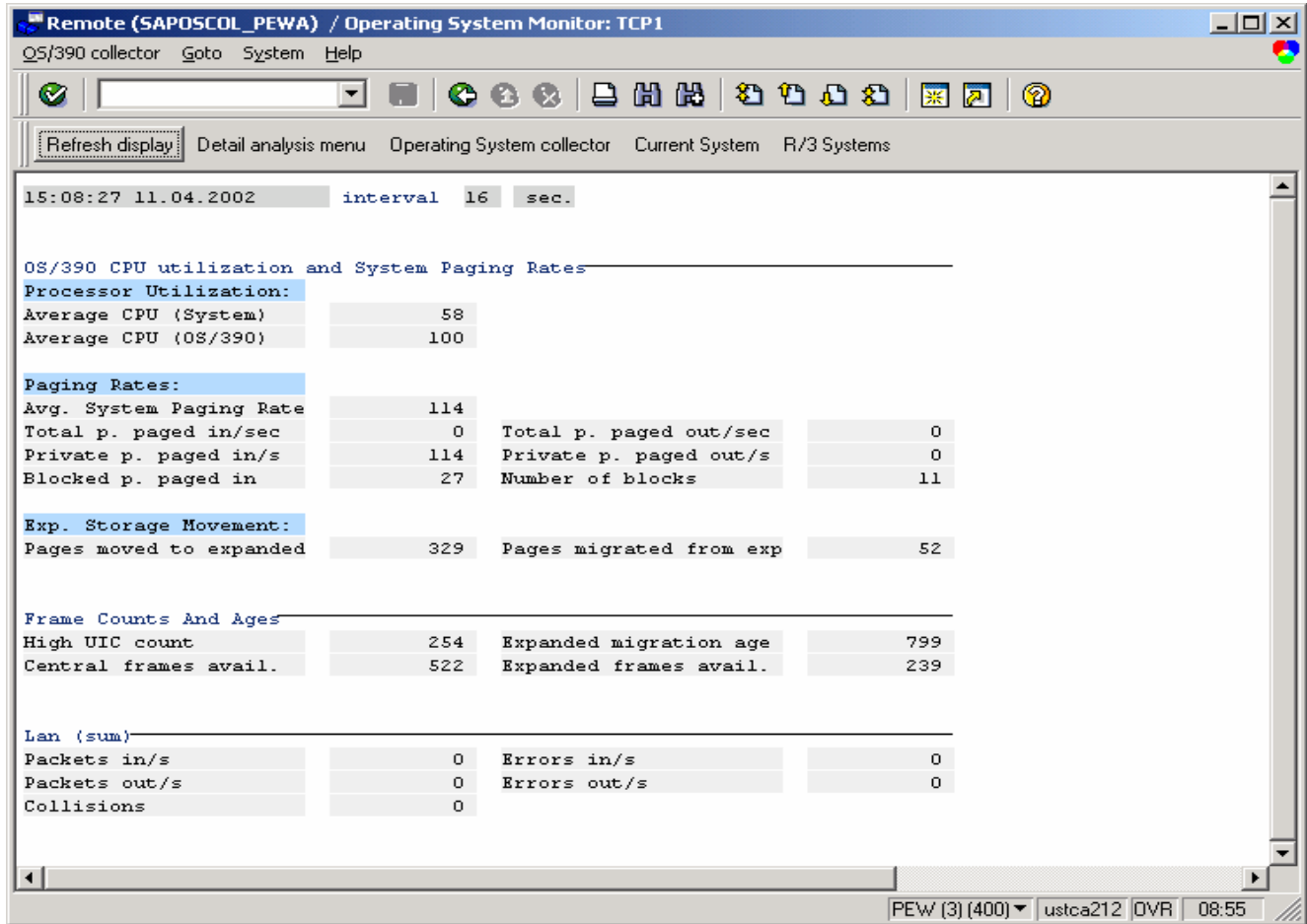


Figure 72: OS07 - overview of DB server performance metrics

OS07 shows that at this moment the LPAR is using 100% of its CPU on the system. Use tools such as RMF I and RMF III to view recent periods, or longer periods.

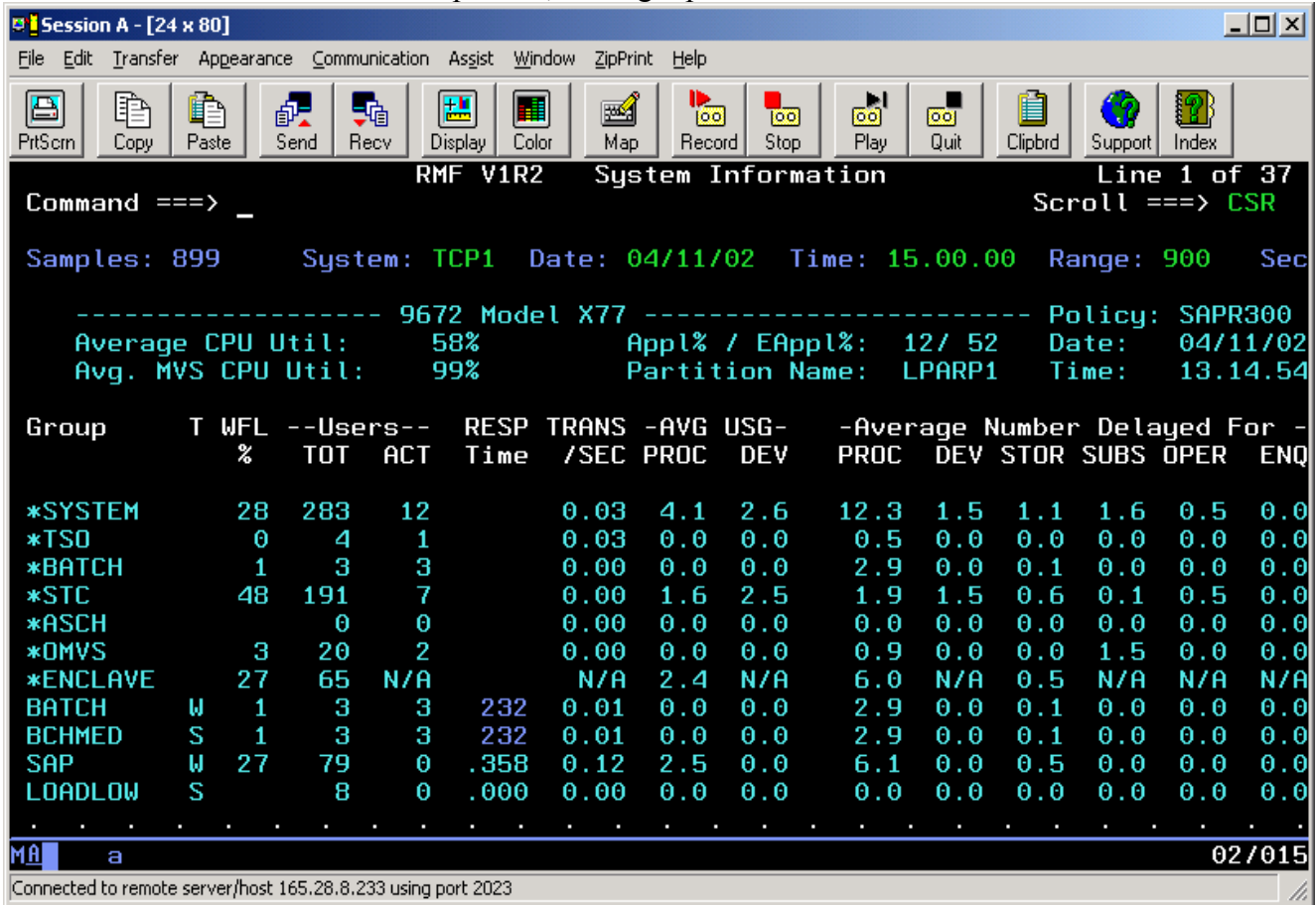


Figure 73: RMF III SYSINFO of 900 second interval

Log into DB server, and use RMF III SYSINFO command, which here shows 99% CPU utilization over a 900 second (“range”) period.

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. 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. Many DB and OS performance problems (bad bufferpool hitrates, high CPU utilization, I/O bottlenecks, etc) can be 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 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 must be separately reviewed on each DB2 subsystem.

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 to do performance analysis on an SAP system on DB2 for OS/390. 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 IFCID 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
- High getpages and low rows processed
- Long statement average elapsed time

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.

When searching the statement cache for inefficient SQL, it is helpful to sort the statement entries by total getpages, total rows processed, or total elapsed 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.

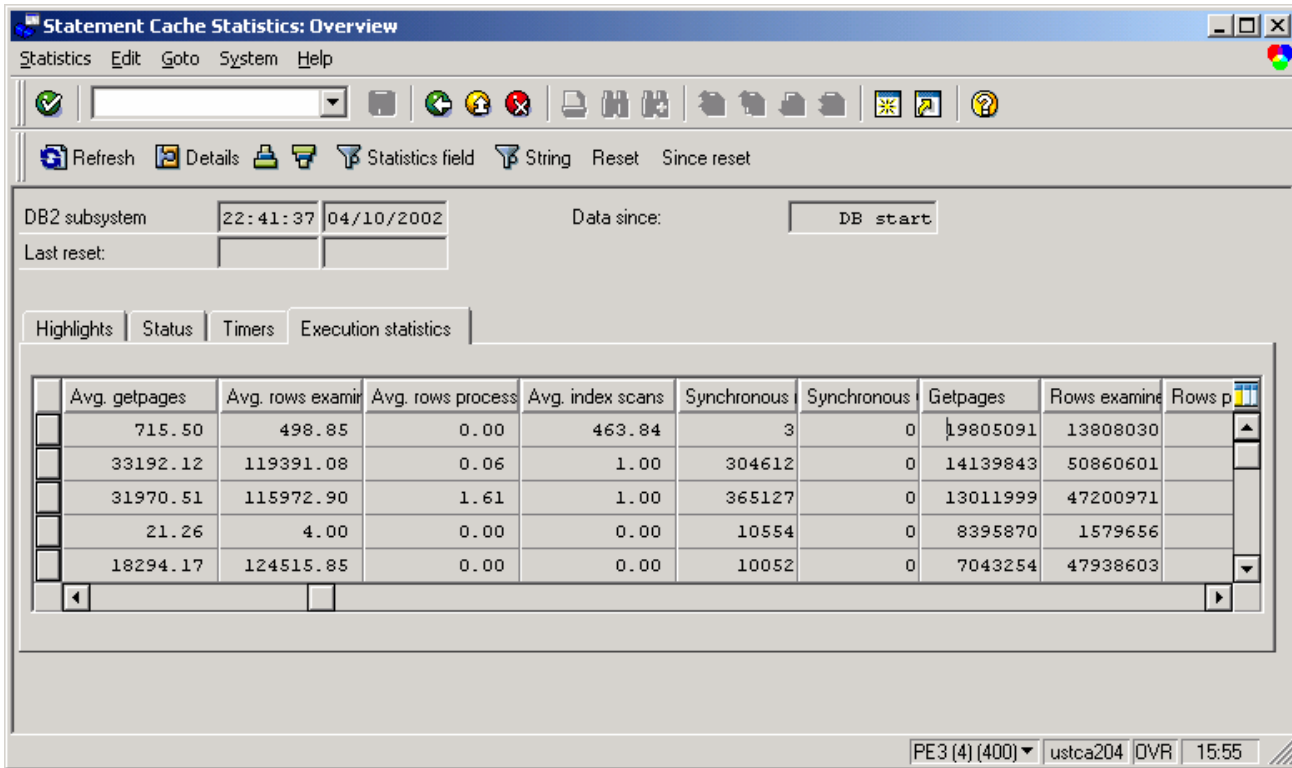


Figure 74: ST04 cached statement statistics sorted by getpages – execution statistics

The execution statistics tab, shown in Figure 74, is used to see per-execution statistics (which are displayed in the “Avg” columns) as well as statistics summed for all executions.



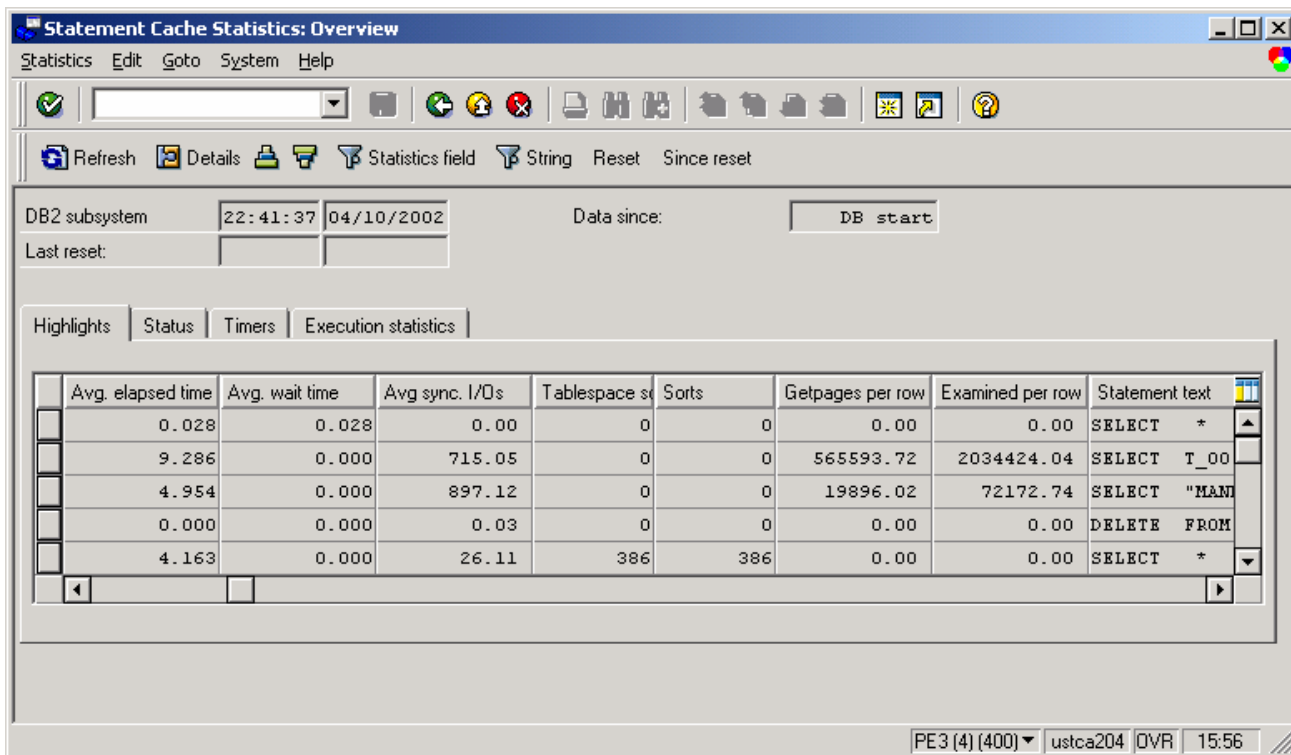


Figure 75: ST04 cached statement statistics sorted by getpages – highlights

The “Highlights” tab, seen in Figure 75, shows several key indicators for SQL problems. These are explained more fully in the following sections.

### 8.3.1. 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 range predicate to not be indexable
- Index screening, where there are gaps in matching index columns from the predicate columns

**In cases where a statement never returns a result, “Getpages per row processed” in “Highlights” 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 per row processed” in “Highlights” is 0, check the “execution statistics”, and look at “Avg. getpages”, which is a per-execution counter. Compare the first row of the statement display in Figure 74 and Figure 75 for an example.**

### 8.3.2. 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 bases on 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
- Predicates contain range predicates, which causes columns in the index to the right of the range predicate to not be indexable

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 per row processed” in “Highlights” 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 “rows examined per row processed” in “Highlights” is 0, check the “execution statistics”, and look at “Avg. rows examined”, which is a per-execution counter.**

### 8.3.3. 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:

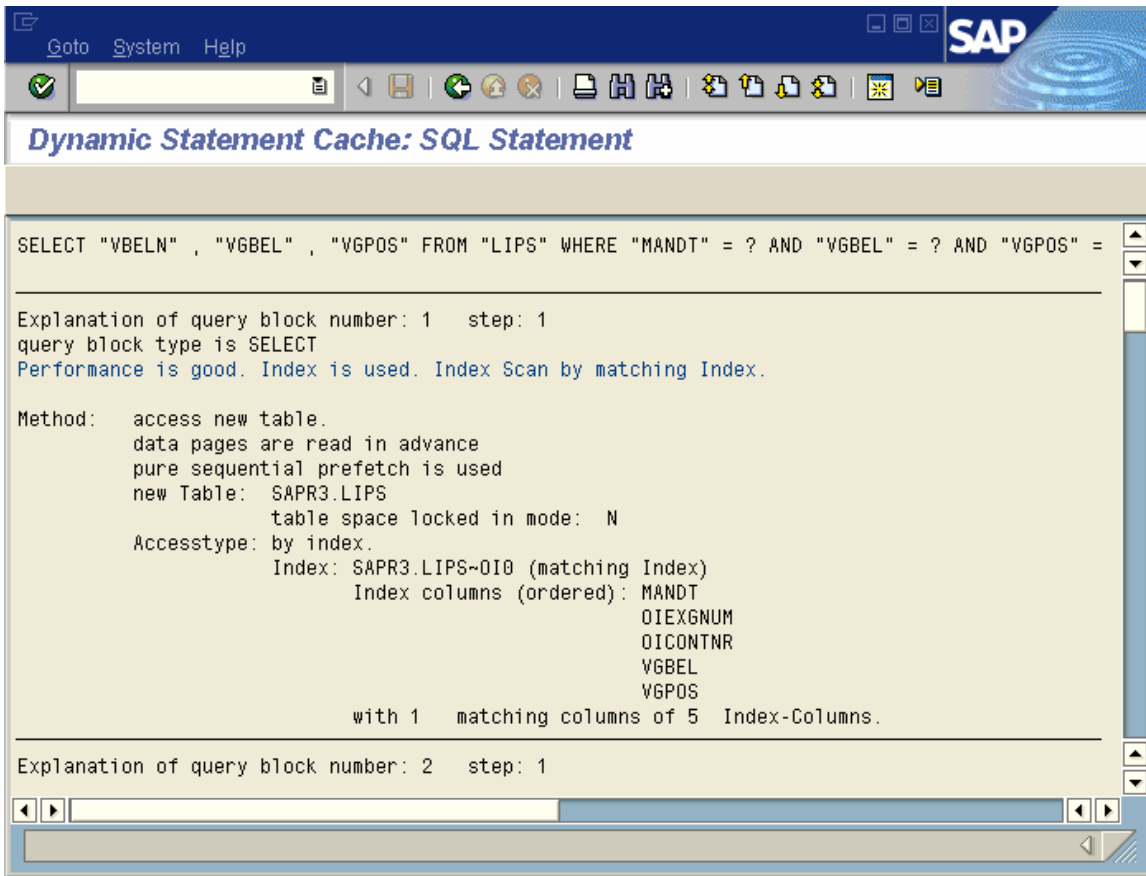
- I/O constraint on disk where the table or index resides
- Logical row lock contention, which is seen with change SQL and “select for update”.
- Inefficient SQL as described above
- DB2 contention on page latches

## 8.4. *Examples of searching for slow or inefficient SQL*

### 8.4.1. Using SE11 “where used”

With DB2 V7, the statement cache statistics will be enhanced to contain the name of the program associated with each SQL statement in the cache. For sites with DB2 V5 or V6, the SE11 “where used” must be used to find the program containing the problematic SQL.

This example does shows only the process for using “where used”, and how to limit the range of the search.



**Figure 76: LIPS with index screening**

In this example, we are searching for programs that use the table LIPS. In general, the MANDT selection will not be specified in the ABAP program. By default, an ABAP program reads only the MANDT values of the MANDT that it is executing in. The statement that we will search for will most likely have the predicate “where VGBEL = and VGPOS =”.

Use SE11 to run “where used”. Where used is the icon with 3 arrows.

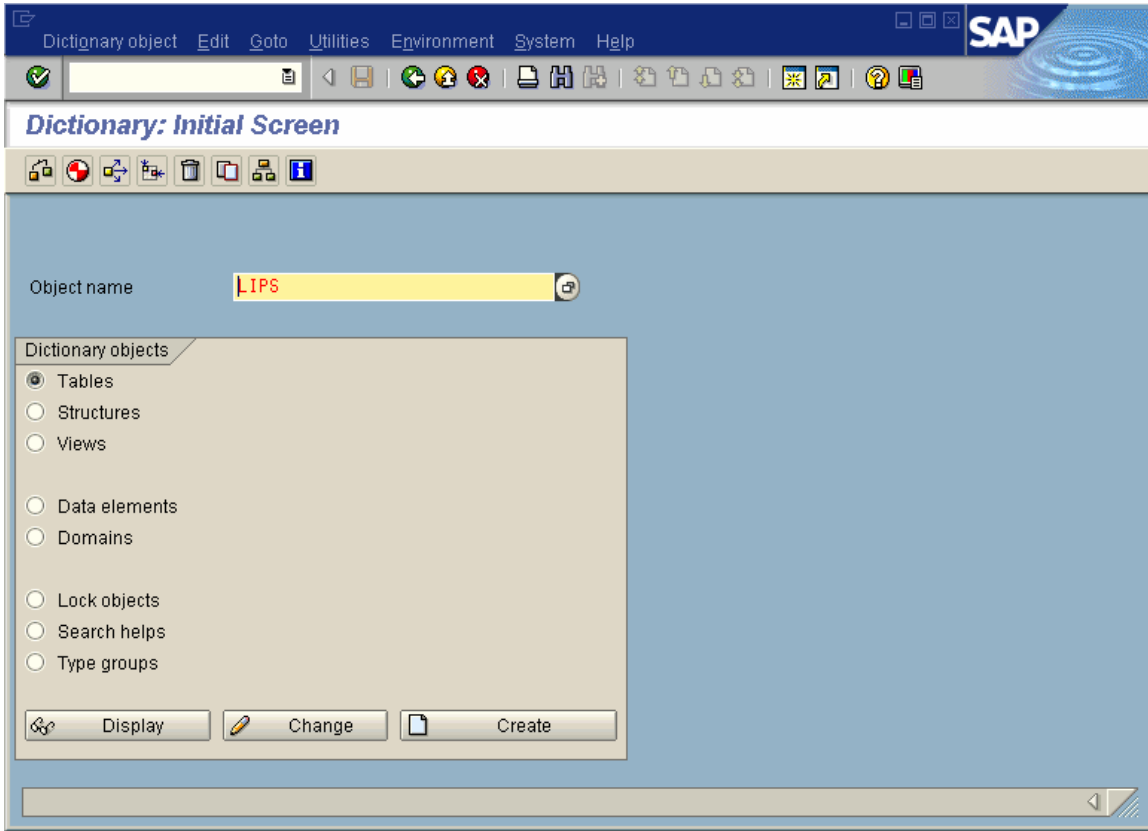
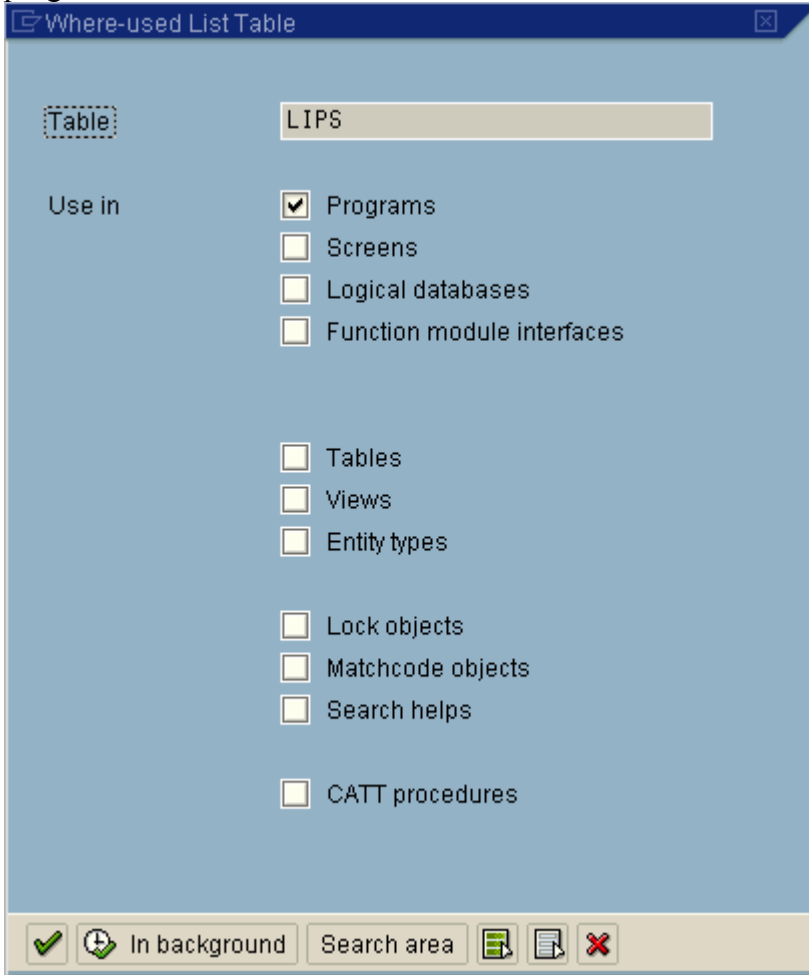


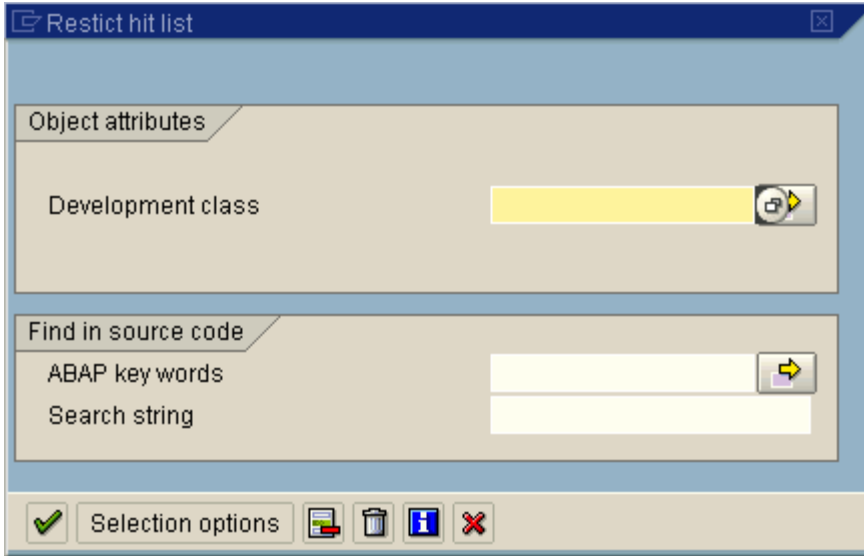
Figure 77: SE11 to initiate “where used”

The “where used” popup will ask for the range of objects to be searched. Usually, searching the programs is sufficient.



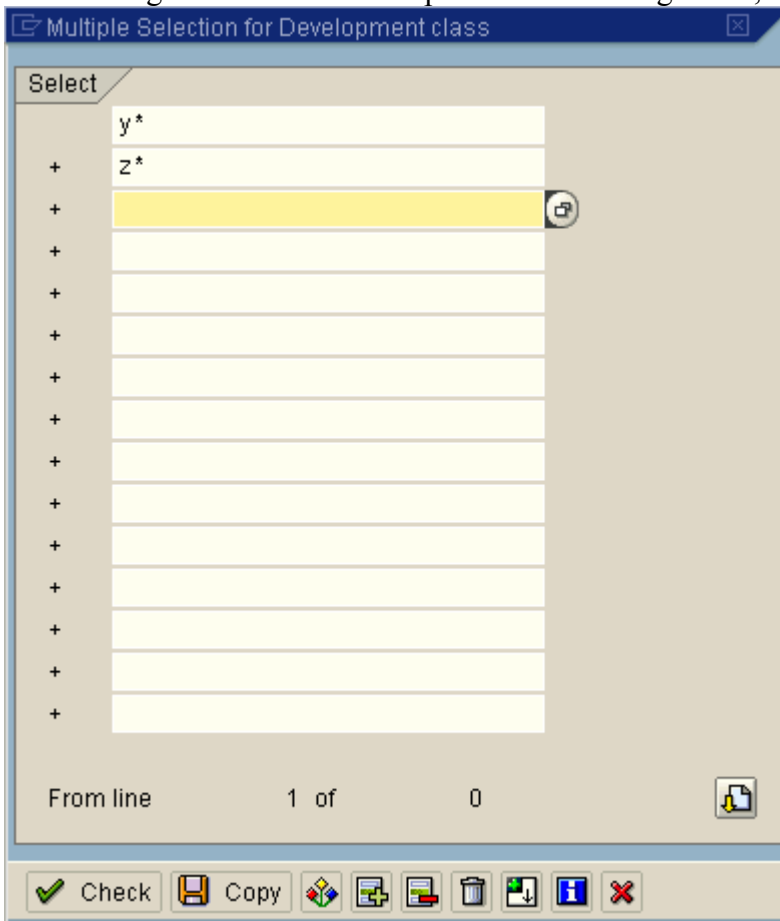
**Figure 78: SE11 “where used” object selection**

In addition to specifying objects to be searched on the screen above, by pressing “Search area” you can narrow the range within the objects, selecting, for example, only customer development classes – Z\* and Y\*. In general, it is a good practice to first search for SQL problems in custom code, before searching SAP standard code. The odds are high that problem SQL is custom written.



**Figure 79: SE11 “Search area” popup**

Press the right arrow on “Development class” in Figure 79, to specify more than one entry.



**Figure 80: SE11 set development class in search area**

Press Copy, enter, enter to perform the search.

You will get the hit-list. Press “select all” in the list, then “detail view lines” (the green plus sign) to expand the list.

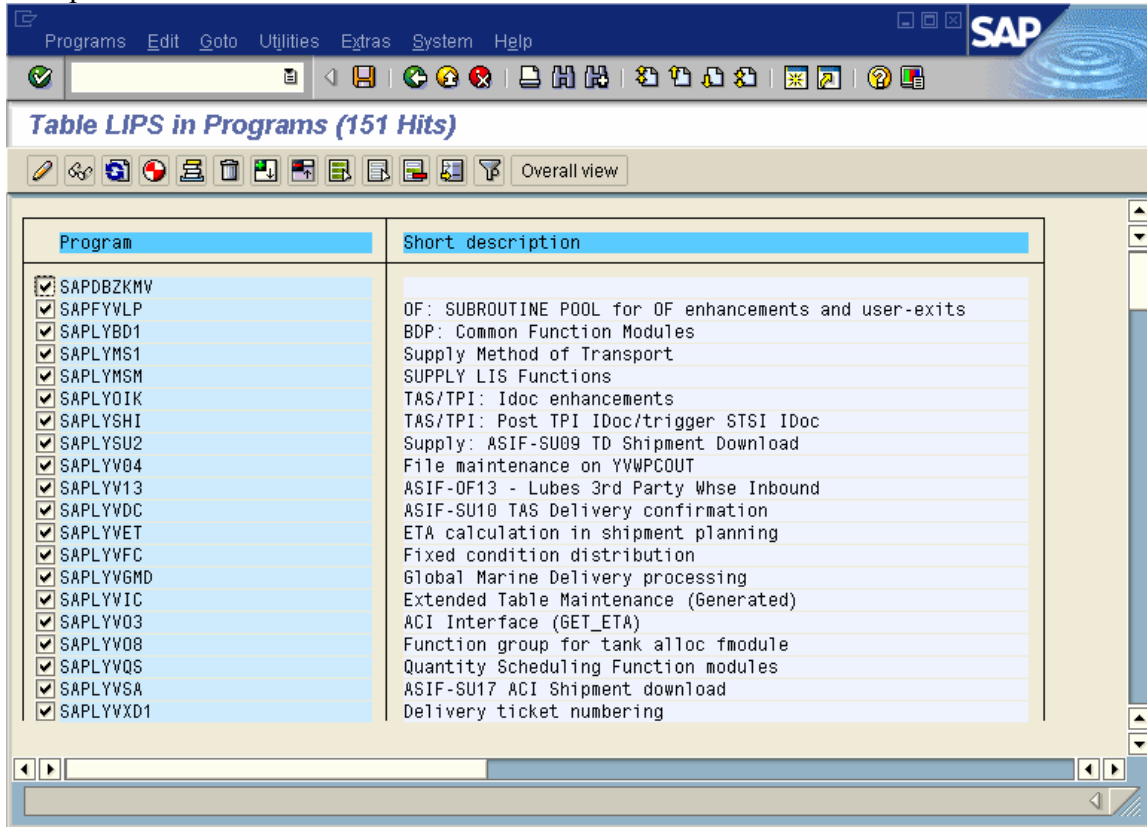


Figure 81: SE11 hit list

The expanded list will look like this.

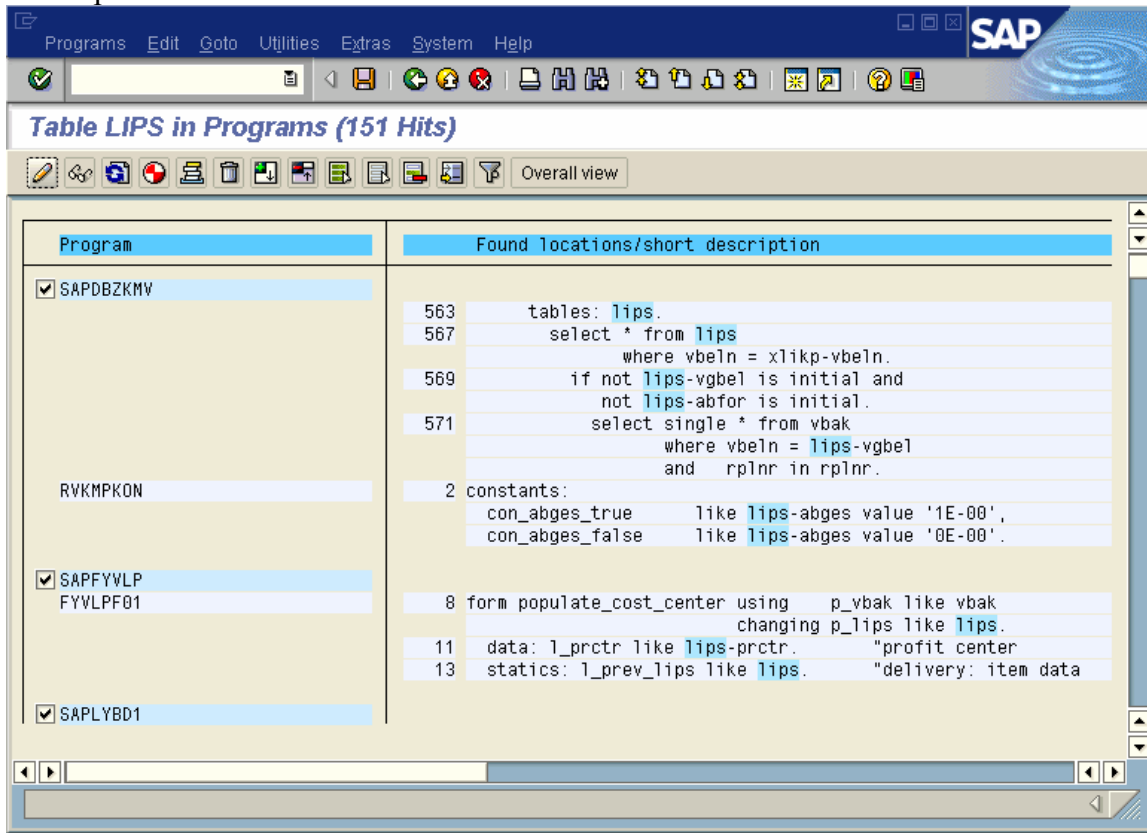


Figure 82: SE11 where used expanded hit list

Search through the expanded hit list using find.

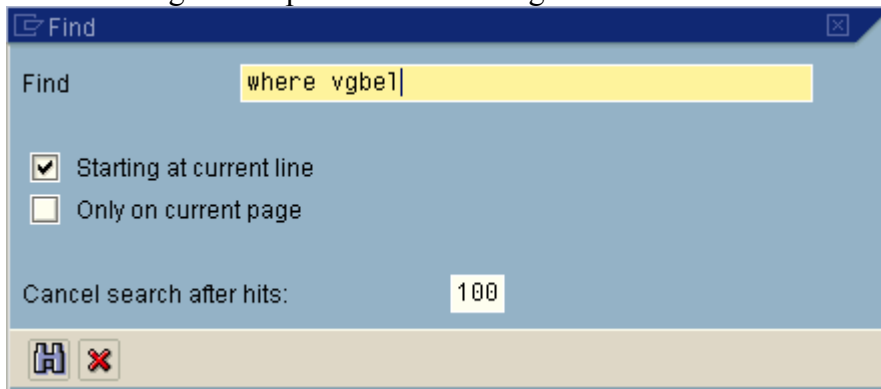


Figure 83: SE11 find to locate search string

We use “where vgbel”, since that should help to narrow the range. Use either result columns that were specified in the select (select xxx yyy zzz) or the predicate columns (where aaa = and bbb =) to help narrow the search. In cases where the SQL is dynamically generated in the ABAP, only the table name will be found by the search.



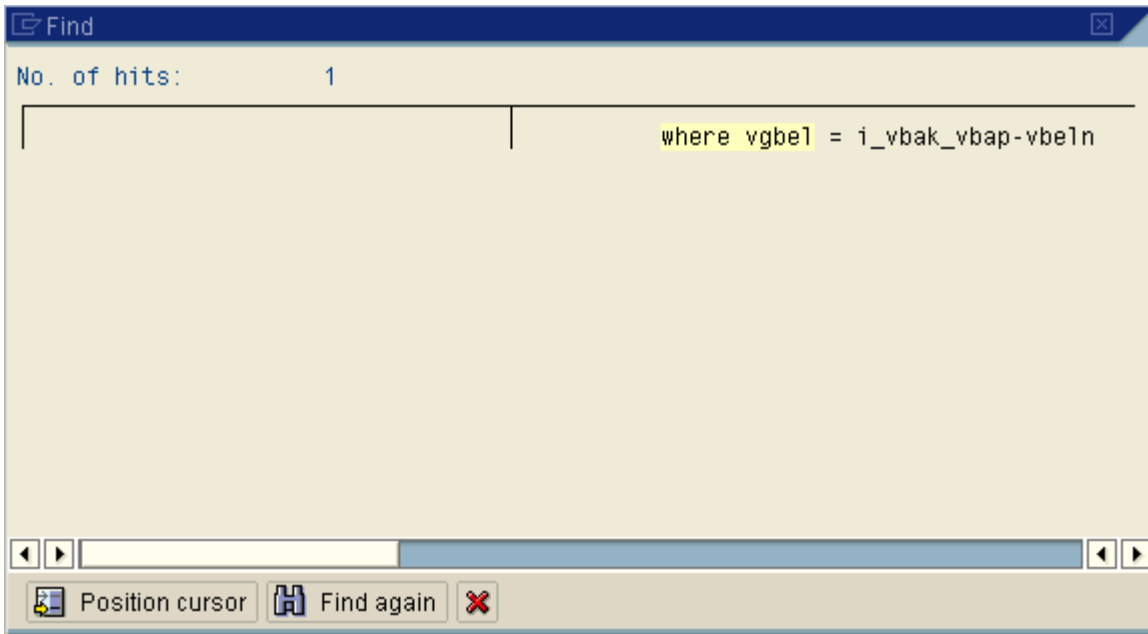


Figure 84: SE11 found lines

Drill into the line, to see the text of the statement.

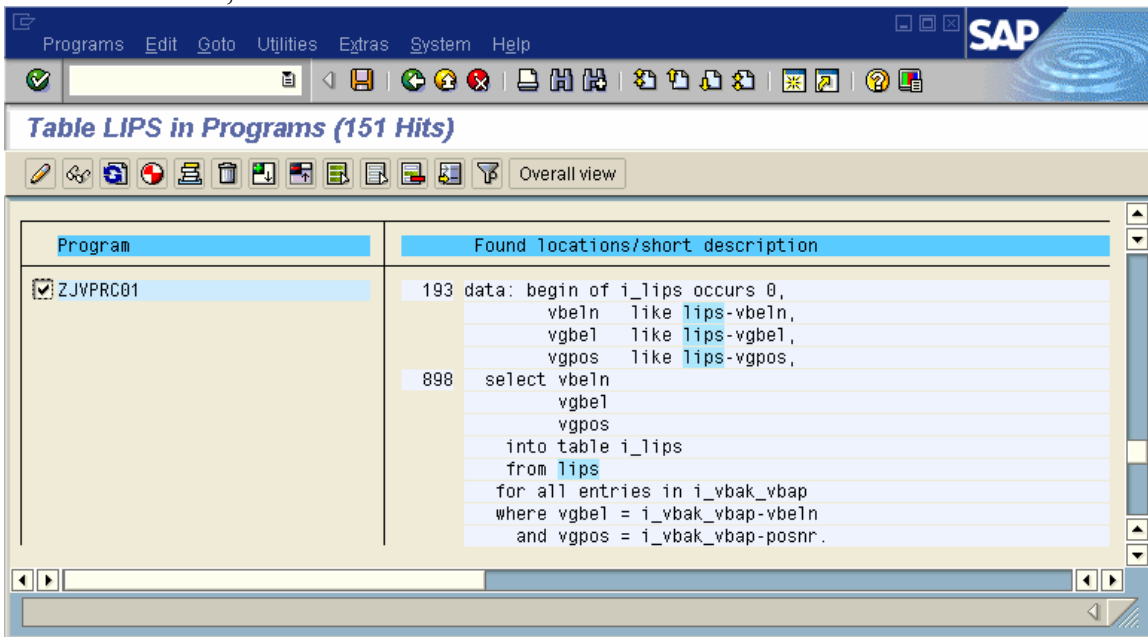


Figure 85: SE11 found program

This will generate a UNION ALL statement in DB2, since there is more than one column specified in the “for all entries”. If there is only one column in the “for all entries”, and IN list will be generated for DB2.

If, after searching the Y\* and Z\* development class, the problem has not been found, leave off the “search range” specification, and all programs will be searched.

Keep in mind that the statements in the ABAP are templates, and if no value is specified for a predicate at runtime, the SQL in DB2 will look different than the SQL in the ABAP. The ABAP can contain “where conditions” that are not in the DB2 SQL being executed. This occurs most commonly in reporting SQL, where the user gets a screen to fill in with selection criteria on many different columns, such as material number, factory, distribution channel, customer, etc. In the ABAP source code, all columns would be present, but only the columns for which the user specified data would be present in the DB2 SQL being executed.

### 8.4.2. Predicates do not match available indexes

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

The most common problem with inefficient SQL is when the 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.

After sorting the ST04 statement cache by getpages, this statement was near the top of the list. We want to determine whether it is efficiently coded, or if it can be improved. Select a statement from the list of cached statements, and press details or drill into the statement.

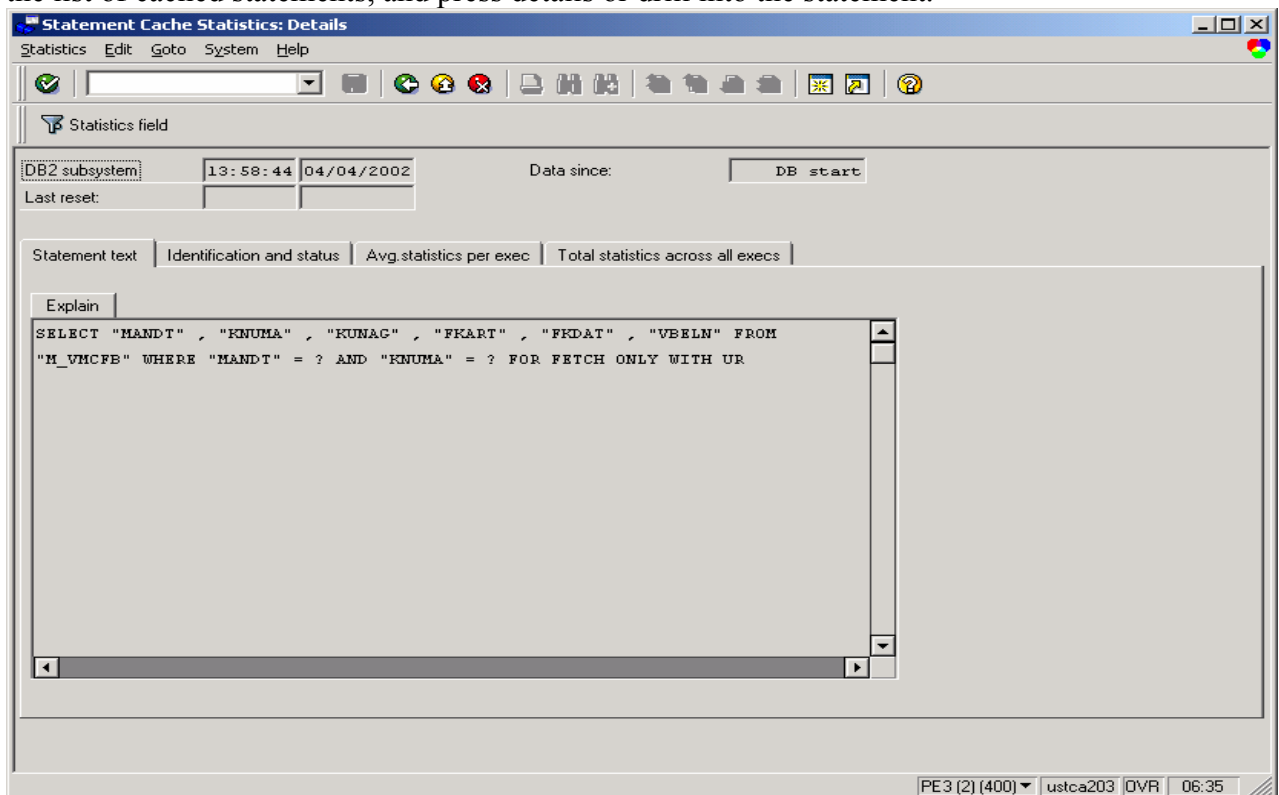


Figure 86: “details” display of M\_VMCFB statement from ST04 cached statement

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 30,993 getpages per execution, and it processes (returns) less than one row (0.12) per execution. An efficiently indexed R/3 statement usually needs just a few getpages per row.

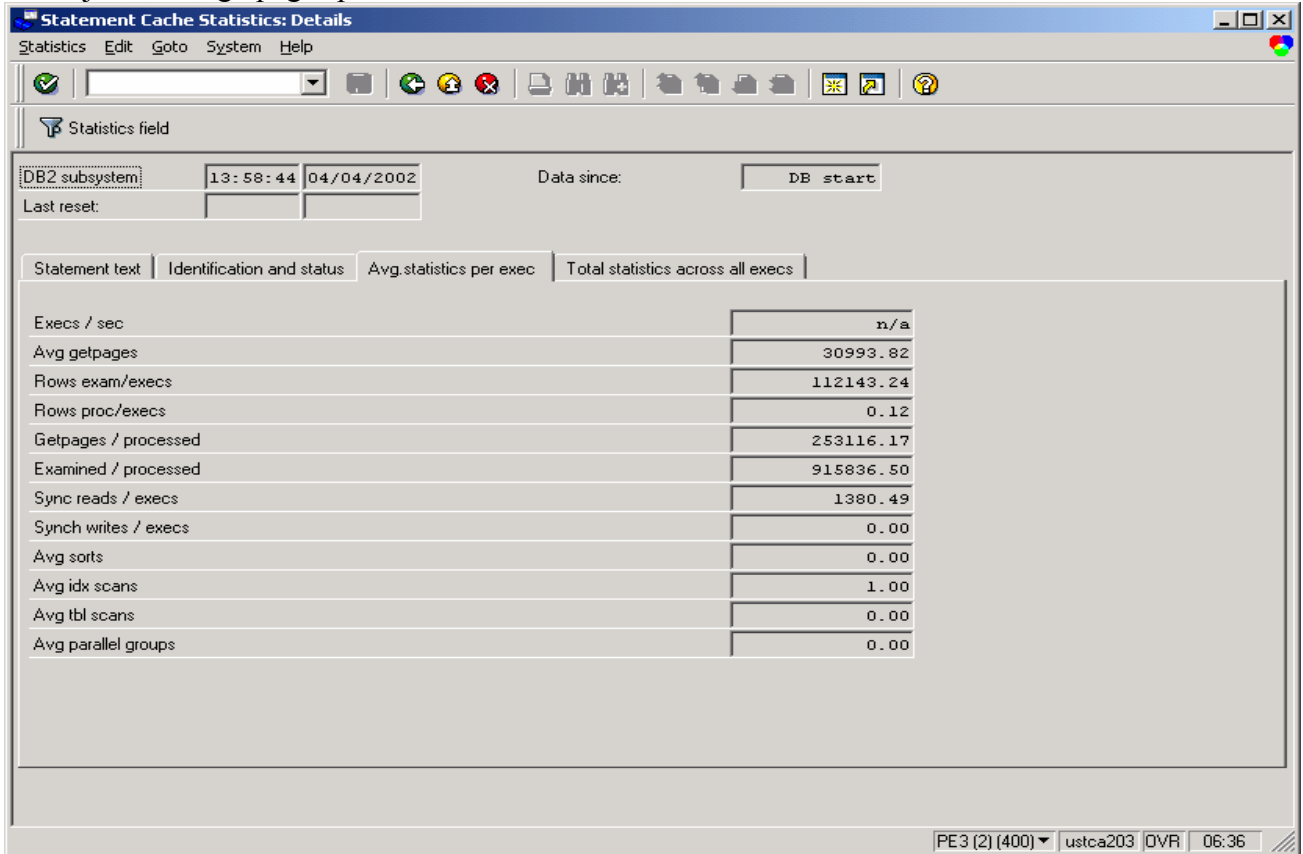
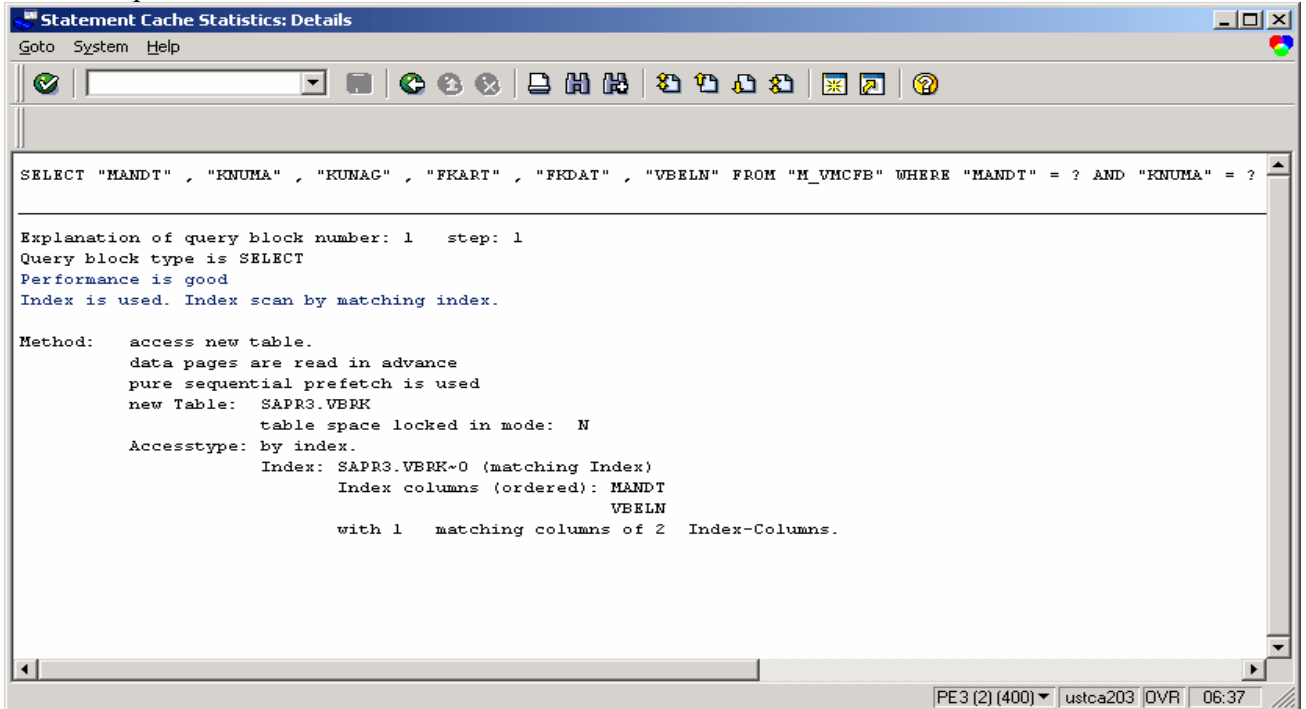


Figure 87: ST04 cache “details” display of execution statistics

Next, from the “statement text” tab in “details”, explain the statement to check the access path being used. The explain shows that M\_VMCFB is a view on VBRK. MANDT is the only matching column on the index. Since MANDT does not filter the data (there is generally one productive client that contains almost all rows) this is not a good index choice. Note that the KNUMA predicate is not used in the index access.



```

Statement Cache Statistics: Details
Goto System Help

SELECT "MANDT" , "KNUMA" , "KUNAG" , "FKART" , "FKDAT" , "VBELN" FROM "M_VMCFB" WHERE "MANDT" = ? AND "KNUMA" = ?

Explanation of query block number: 1  step: 1
Query block type is SELECT
Performance is good
Index is used. Index scan by matching index.

Method:  access new table.
         data pages are read in advance
         pure sequential prefetch is used
new Table:  SAPR3.VBRK
         table space locked in mode:  N
Accessstype:  by index.
             Index:  SAPR3.VBRK-0 (matching Index)
                 Index columns (ordered):  MANDT
                                           VBELN
                 with 1  matching columns of 2  Index-Columns.
  
```

PE3 (2) (400) | ustca203 OVR | 06:37

**Figure 88: Explained statement from ST04 cached statement details**

Use the ST04 DB2 catalog browser to check to see if KNUMA is in another index on VBRK.

The screenshot shows the DB2 Catalog Browser interface. The 'Enter SQL statement' field contains the following query:

```

SELECT A.IXCREATOR, A.IXNAME, A.COLNAME, A.COLSEQ, A.ORDERING,
       A.COLNO, HEX(A.COLNO) AS HEX_COLNO
FROM SYSIBM.SYSKEYS A
     ,SYSIBM.SYSINDEXES B
WHERE A.IXCREATOR = B.CREATOR
AND   A.IXNAME    = B.NAME
AND   B.CREATOR  = 'SAPR3'
AND   B.TENAME   IN ('VBRK')
ORDER BY A.IXCREATOR, A.IXNAME, A.COLSEQ
    
```

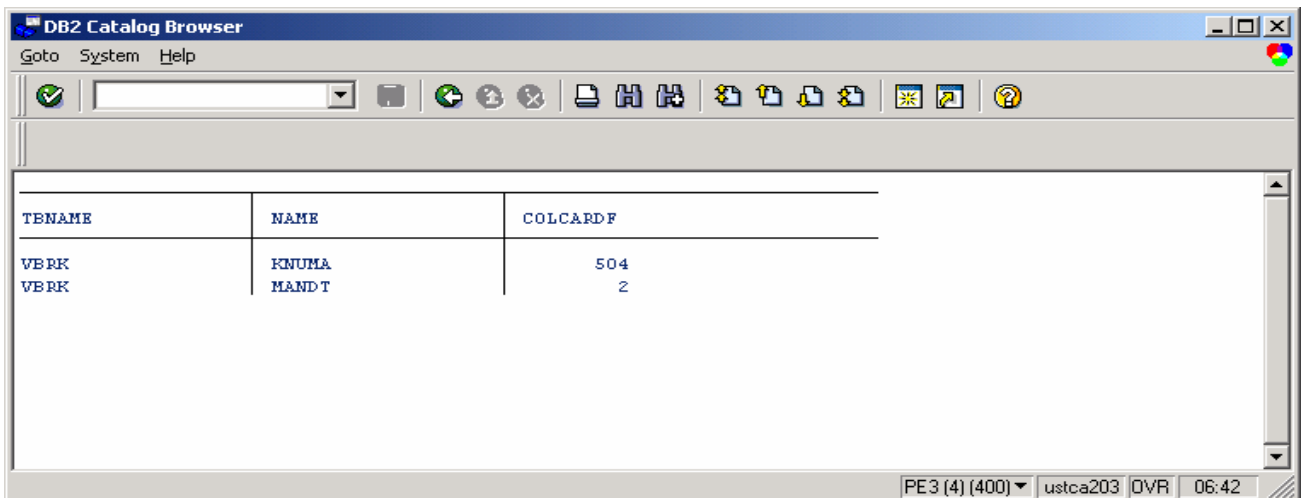
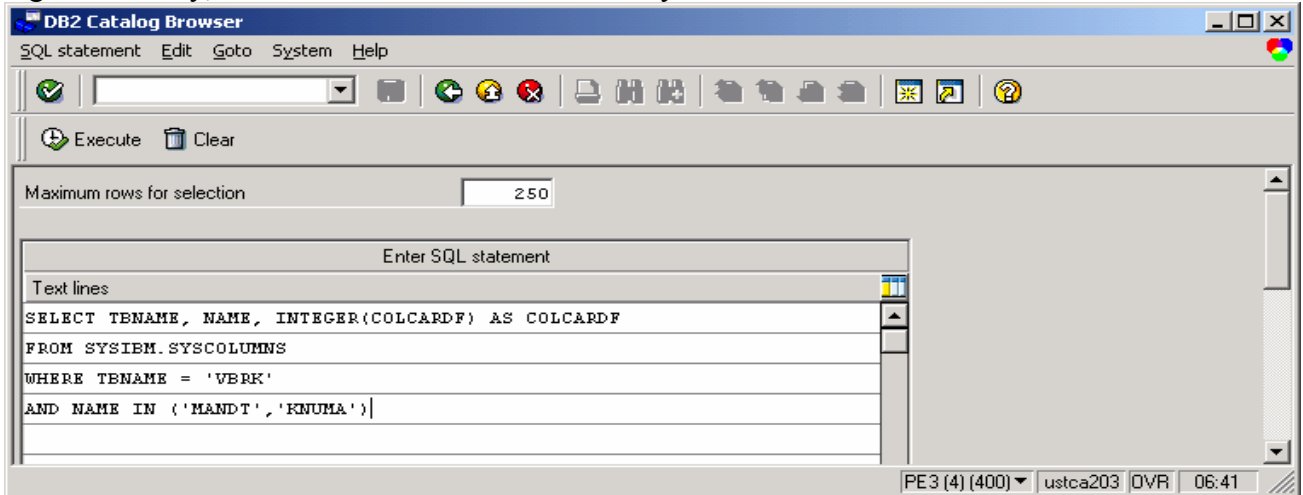
The status bar at the bottom indicates: PE3 (4) (400) | ustca203 | OVR | 06:39

KNUMA is not in any index, so it cannot be used.

The screenshot shows the results of the SQL query in a table format. The status bar at the bottom indicates: PE3 (4) (400) | ustca203 | OVR | 06:40

IXCREATOR	IXNAME	COLNAME	COLSEQ	ORDERING	COLNO	HEX_COLNO
SAPR3	VBRK~LOC	MANDT	1	A	1	0001
SAPR3	VBRK~LOC	LCNUM	2	A	86	0056
SAPR3	VBRK~0	MANDT	1	A	1	0001
SAPR3	VBRK~0	VBELN	2	A	2	0002

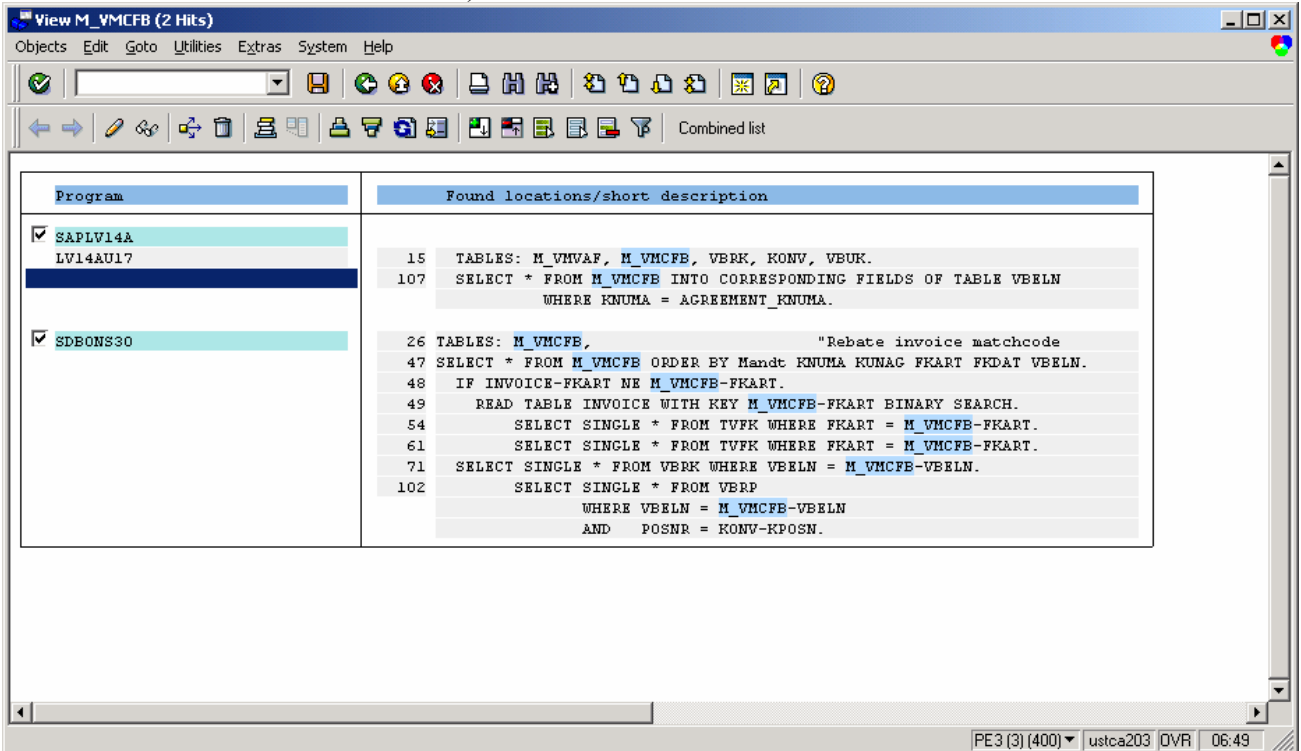
Now check if KNUMA has high cardinality. Cardinality is the number of unique values. If it had high cardinality, it would make a more efficient way to access the table with an index.



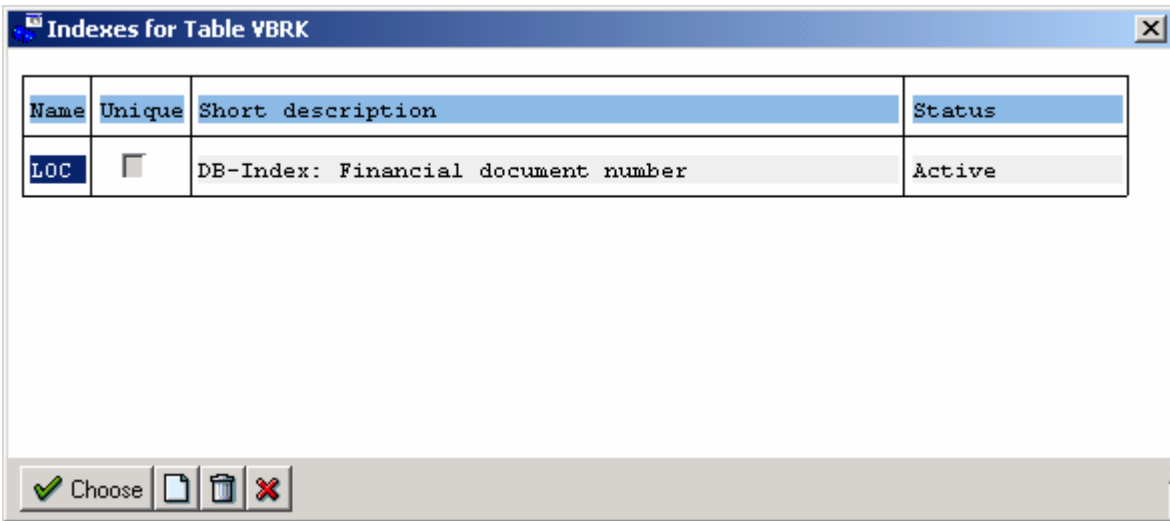
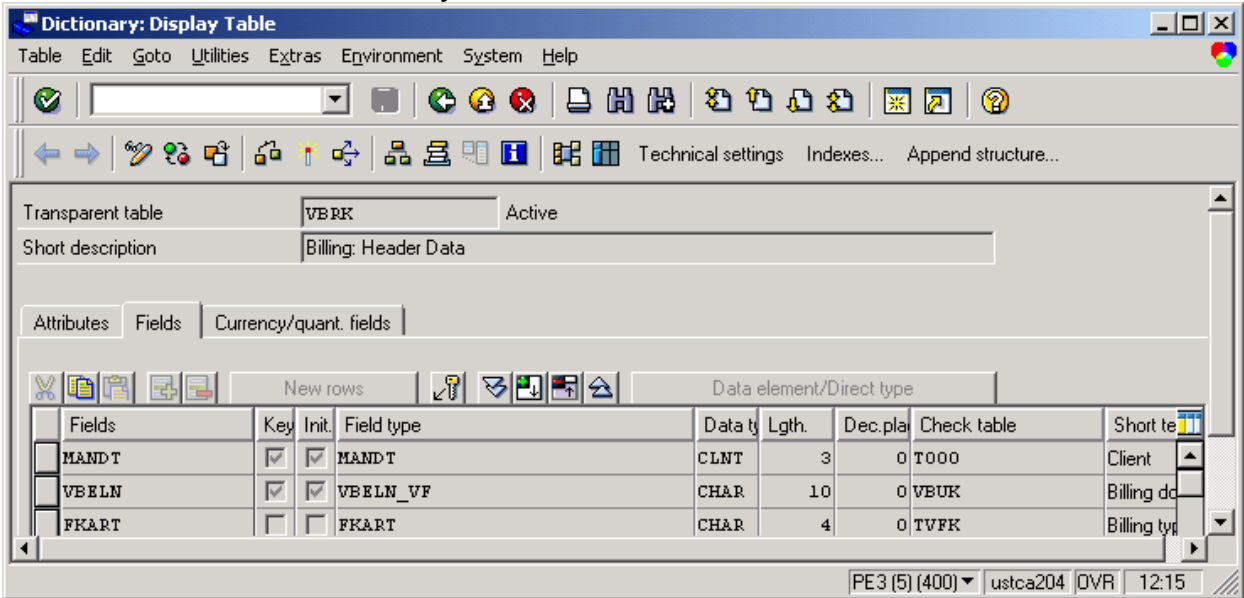
**Figure 89: VBRK column cardinality statistics**

In Figure 89 we see that KNUMA has 504 unique values, so it looks like it will help to filter better than current access path, which uses only MANDT (two unique values) as filter. (The ST04 statistics, which show less than one row processed per execution, demonstrate that KNUMA will be an effective index.)

Next, try to find the culprit. Use SE11 “where used” on M\_VMCFB. After finding and expanding the hit list, note that it is an SAP program issuing the statement. The name of the program starts with an S, so is in the SAP namespace, not customer Z\* & Y\* namespace. (Recall that SAPxZ\* and SAPxY\* contain customer code.)

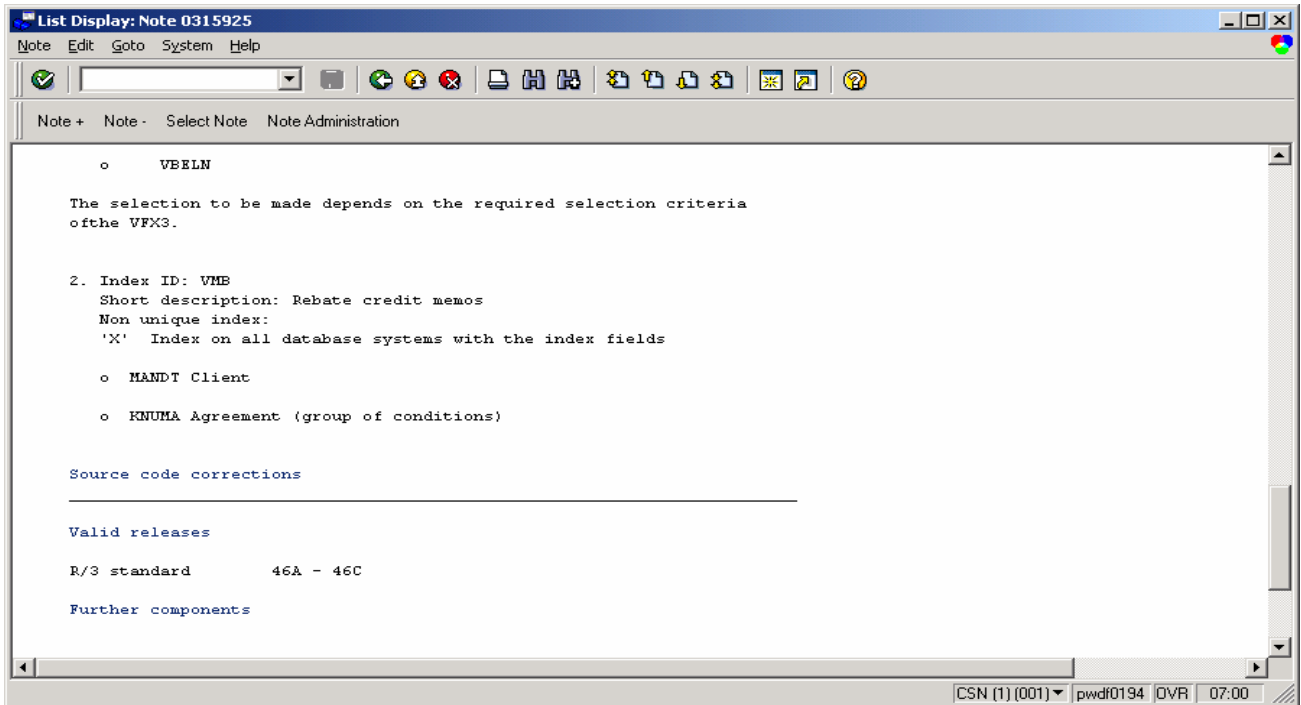


Use SE11 > display, to display the VBRK table. Then select “Indexes” to see if there a secondary index defined in the data dictionary that could be activated.



No other indexes, there is just one secondary index on the table. (SE11 “Indexes” does not display the primary index.) Since it is an SAP program, search OSS for SAPnotes related to the M\_VMCFB view and the table VBRK.





We find a SAPnote that suggests adding another index.

Since the program is an SAP program, the first action is to search for indexes defined in the data dictionary, and then search SAPnotes. There are many indexes that are not part of standard SAP, but which are recommended by SAP to solve specific problems.

### 8.4.3. Incorrect use of SAP tables

Here is some background to help to understand this sample problem. In SAP, header tables often have many indexes, to allow documents (sales orders, purchase orders, etc) to be found in many different ways. Line-item detail tables are generally indexed only by document number. Header tables often end in K, and document tables often end in P. SE11 shows the description of the table, which will specify if it is header or line item. Relationships between SAP sales documents (e.g. which delivery was created for an order, or which order triggered an invoice) are contained in VBFA.

Here is an SQL statement that is *number three* in the ST04 statement cache list sorted by total getpages. Since it is near to the top of the list, it is having a significant impact on the overall system resource usage during the interval of monitoring. “Getpages per row” (415,360) and “examined per row” (1,489,213) are both huge.

Statement Cache Statistics: Overview

DB2 subsystem: 18:05:08 04/03/2002 Data since: DB start

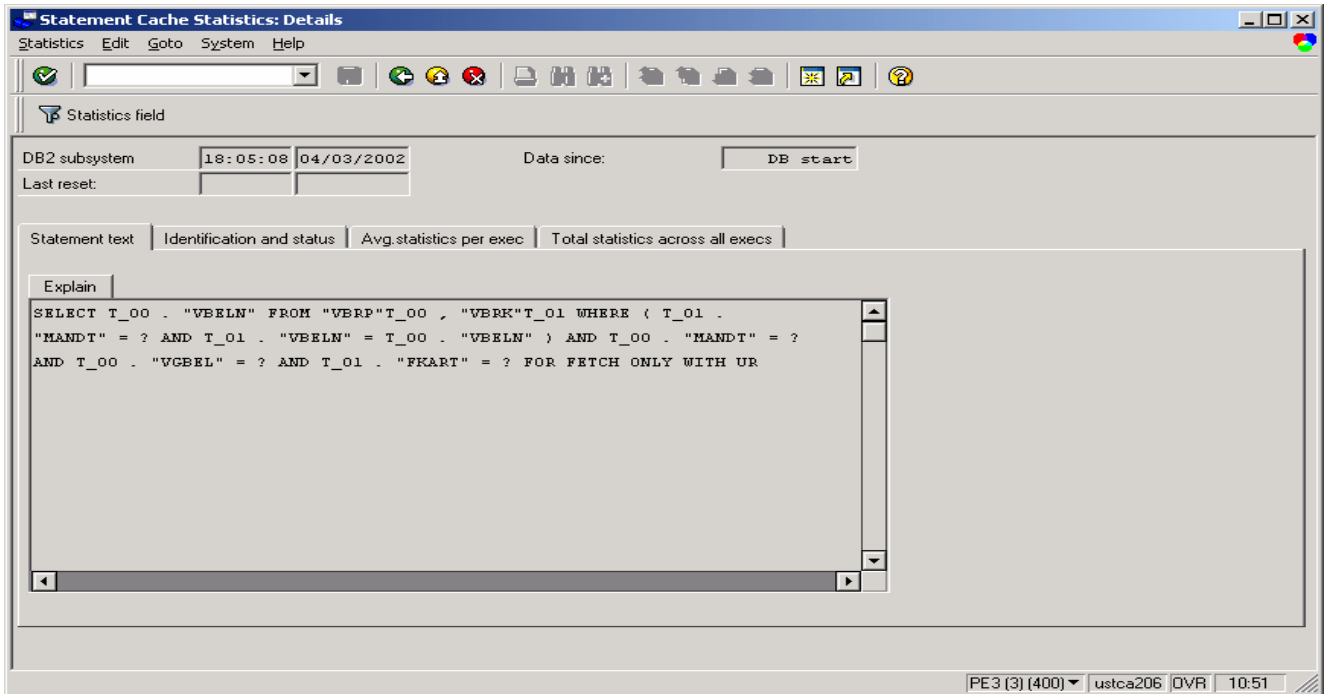
Last reset:

Highlights | Status | Timers | Execution statistics

Executions	Avg. elapsed time	Avg. wait time	Avg sync. I/Os	Tablespace s	Sorts	Getpages per row	Examined per row	Statement text
213	13.024	0.000	2384.73	0	0	30917.64	88963.89	SELECT * FR
3569	0.070	0.000	0.01	3570	0	0.00	0.00	SELECT * FR
38	19.847	0.000	1018.13	0	0	415360.00	1489213.33	SELECT T_00 .
910	0.029	0.000	0.00	0	0	0.00	0.00	SELECT * FR
9	7.386	0.000	1192.56	0	0	0.00	0.00	SELECT "MANDT"
12479	0.012	0.000	1.71	0	0	3.24	2.01	SELECT * FR
33	0.458	0.000	0.00	33	33	5.67	15.71	SELECT "CJAHR"
1	6:59.355	0.000	59088.00	0	0	2.60	10.97	SELECT * FR
53887	0.000	0.000	0.02	0	0	29.96	43.83	SELECT "ABMNC"
74446	0.001	0.000	0.24	0	0	0.86	1.46	SELECT * FR
12453	0.003	0.000	1.24	0	0	2.44	2.00	SELECT * FR

PE3 (3) (400) | ustca206 | OVR | 10:51

Display the statement, and see that it is a join on VBRK and VBRP. This is not unusual. One often sees a header table (-K) joined to its line item table (-P). Take note that MANDT, VGBEL (in T\_00 -- VBRP) and FKART (in T\_01 --VBRK) are the predicate columns. The tables are joined on MANDT and VBELN. We will need this information when looking for candidate indexes.



**Figure 90: VBRP VBRK join**

The join conditions are in parentheses, and the predicates (FKART and VGBEL) are outside parentheses.

Looking at the per-execution statistics via “details” in the ST04 cached statement statistics, and see that it does about 32,791 getpages per execution, and processes less than one row (0.08) per execution. It is examining over 117,569 rows in order to return less than one. This is very inefficient. In a two-table join, one might normally have 10 getpages per row, if the indexes are efficient.

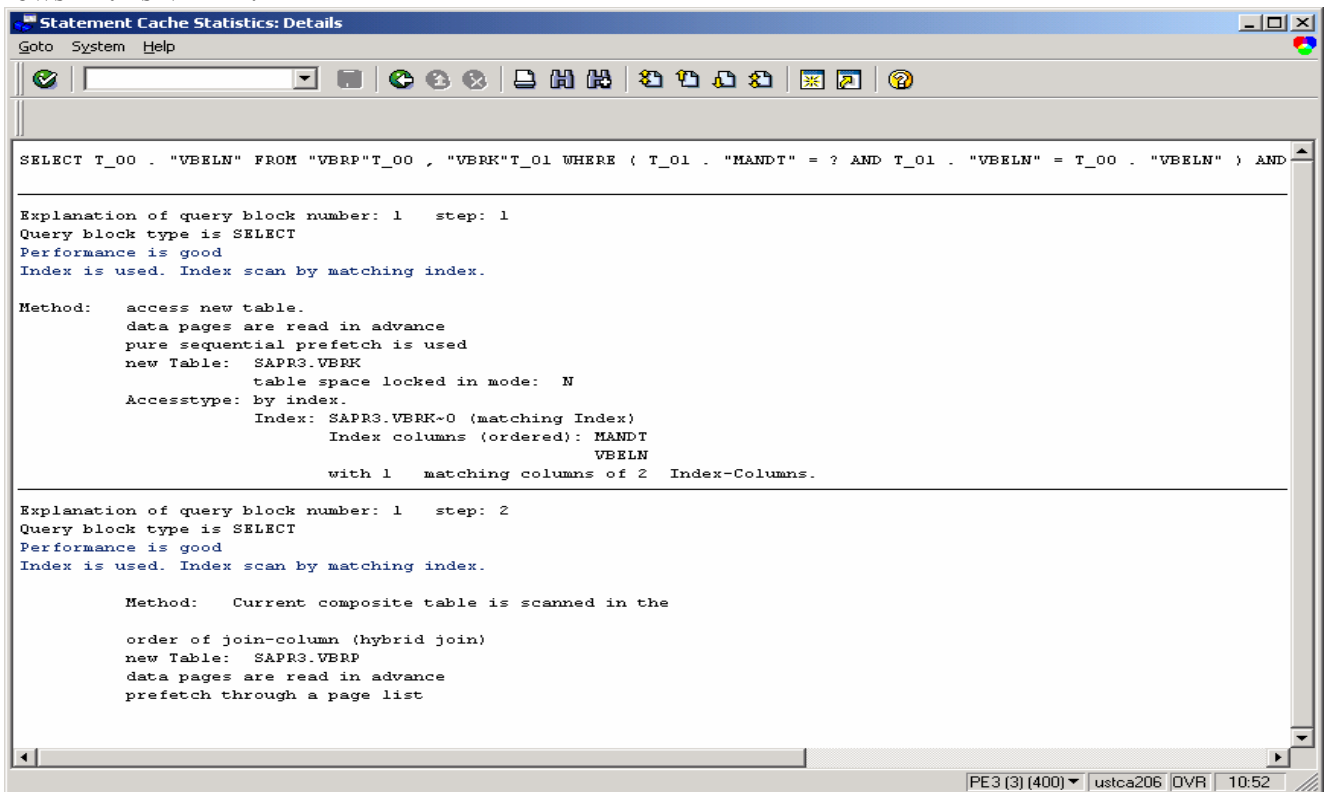
The screenshot shows the 'Statement Cache Statistics: Details' window. At the top, it displays the subsystem 'DB2 subsystem', time '18:05:08', and date '04/03/2002'. The 'Data since' field is set to 'DB start'. Below this, there are tabs for 'Statement text', 'Identification and status', 'Avg. statistics per exec', and 'Total statistics across all execs'. The 'Avg. statistics per exec' tab is active, showing a table of performance metrics.

Metric	Value
Execs / sec	0.00
Avg getpages	32791.58
Rows exam/execs	117569.47
Rows proc/execs	0.08
Getpages / processed	415360.00
Examined / processed	1489213.33
Sync reads / execs	1018.13
Synch writes / execs	0.00
Avg sorts	0.00
Avg idx scans	1.03
Avg tbl scans	0.00
Avg parallel groups	0.00

At the bottom right of the window, the status bar shows 'PE3 (3) (400)', 'ustca206', 'OVR', and '10:52'.

From the “statement text” tab in “details”, explain the statement – it is a hybrid join, where the header table (VBRK) is the driver table. Look at the matching columns on VBRK~0 – only MANDT is matched. This is not good. (In general, be suspicious when you see hybrid join with R/3 – it is usually DB2 trying to make the best of a mismatch between the SQL statement and the indexes available. The same goes for multi-index access, and non-matching index scan, too. Simple transaction SQL does not usually require sophisticated access paths. Nested loop join is the most common join method by far.)

From the predicates in Figure 90, we can see that every row in VBRK will be read from the table, in order to check FKART, and then VBRP can be accessed. That is, every billing document header must be examined. A large company might have hundreds of thousands or millions of rows in this VBRK.



The screenshot shows a window titled "Statement Cache Statistics: Details" with a menu bar (Goto, System, Help) and a toolbar. The main content area displays the following SQL query and execution details:

```

SELECT T_00 . "VBELN" FROM "VBRP" T_00 , "VBRK" T_01 WHERE ( T_01 . "MANDT" = ? AND T_01 . "VBELN" = T_00 . "VBELN" ) AND

```

Explanation of query block number: 1 step: 1  
Query block type is SELECT  
Performance is good  
Index is used. Index scan by matching index.

Method: access new table.  
data pages are read in advance  
pure sequential prefetch is used  
new Table: SAPR3.VBRK  
table space locked in mode: N  
Accesstype: by index.  
Index: SAPR3.VBRK-0 (matching Index)  
Index columns (ordered): MANDT  
VBELN  
with 1 matching columns of 2 Index-Columns.

Explanation of query block number: 1 step: 2  
Query block type is SELECT  
Performance is good  
Index is used. Index scan by matching index.

Method: Current composite table is scanned in the  
order of join-column (hybrid join)  
new Table: SAPR3.VBRP  
data pages are read in advance  
prefetch through a page list

The status bar at the bottom right shows: PE3 (3) (400) ustca206 QVR 10:52

Now, look at the indexes available on VBRK and VBRP. One can use DB02 > detailed analysis (in the tables section at the bottom of the screen) to display the indexes on a table. (The following two screens are new GUI style, while rest of example is “classic GUI”.)

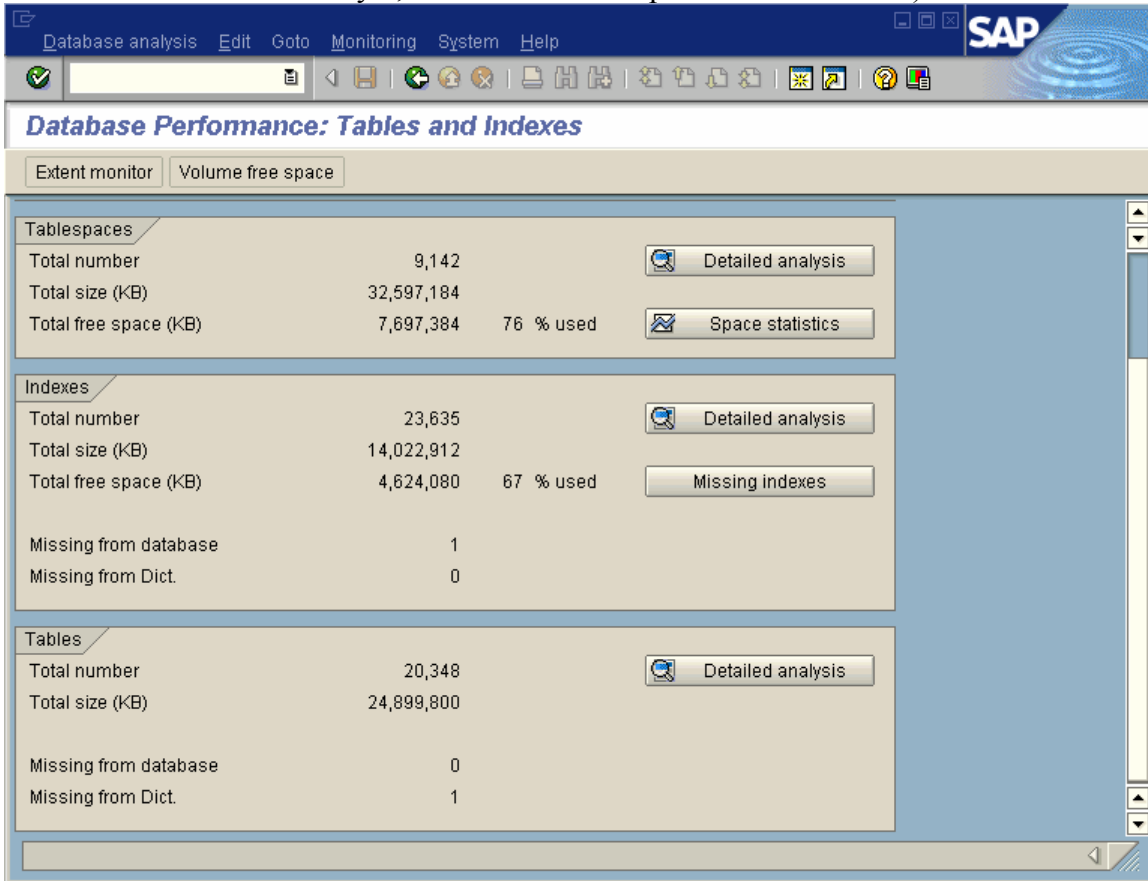


Figure 91: DB02 main screen

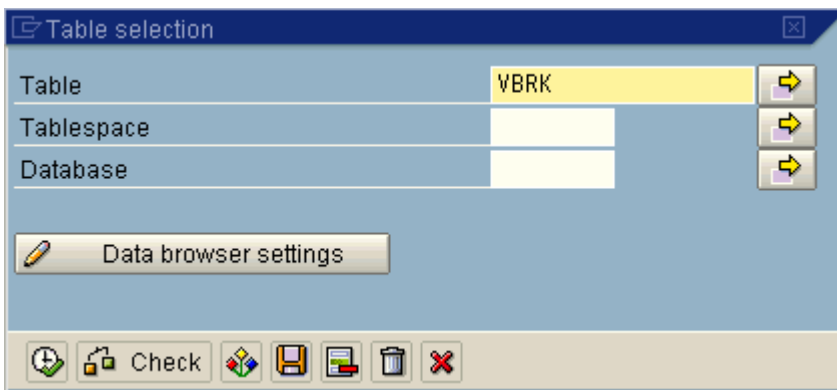


Figure 92: DB02 table detailed analysis - select table

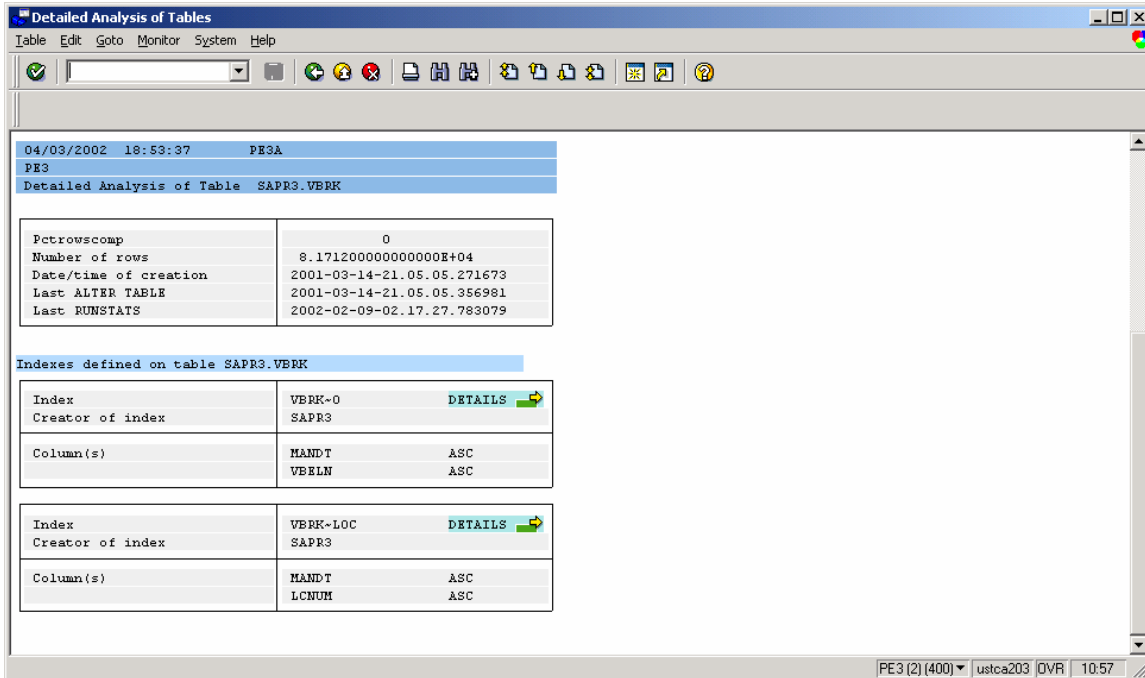


Figure 93: DB02 detailed analysis for VBRK

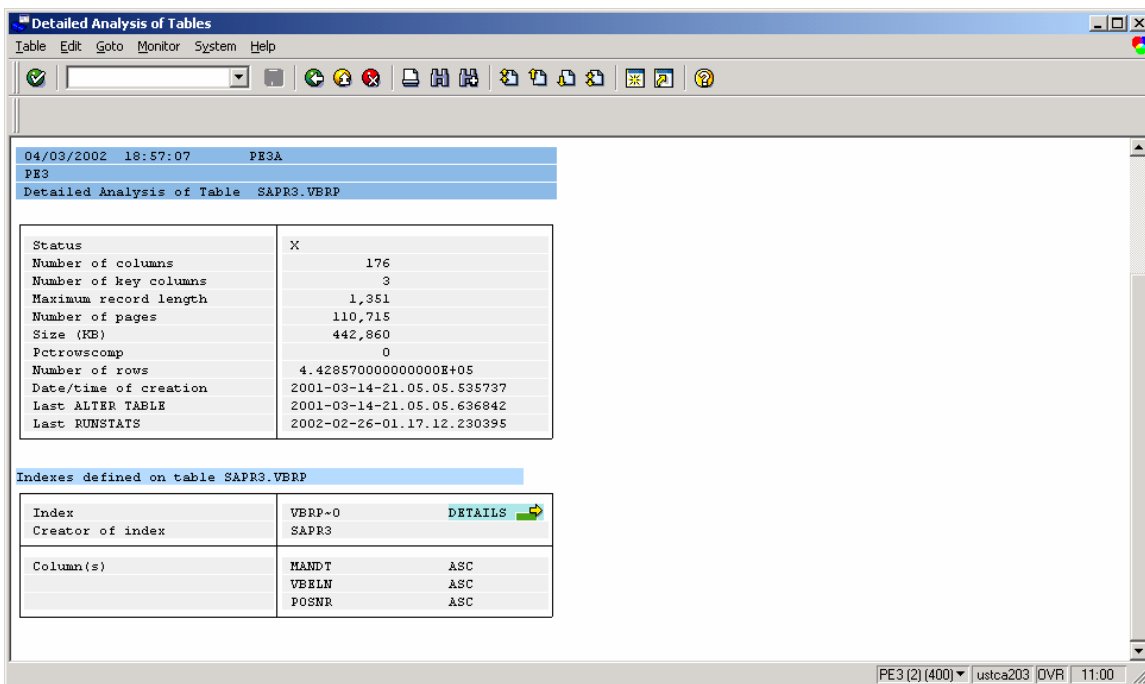
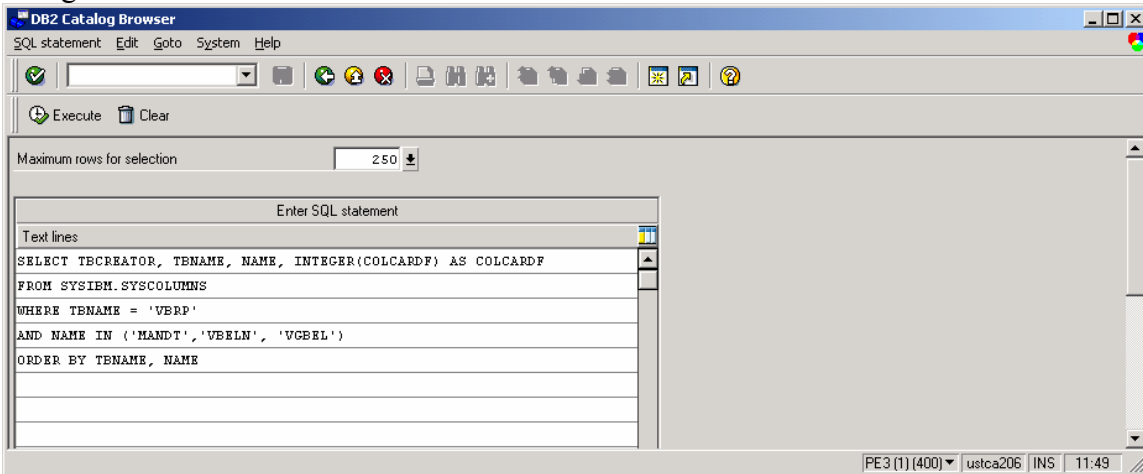


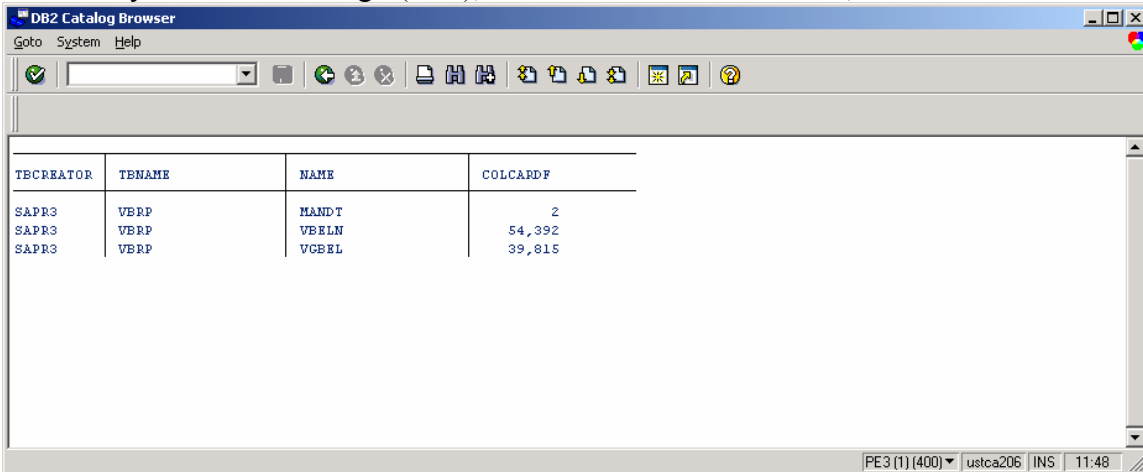
Figure 94: DB02 detailed analysis for VBRP

Checking the two screens above, see that the two predicate columns (FKART and VGBEL) are not indexed on either table. FKART has low cardinality (this check is not shown here). So, we might think that the solution would be an index on VGBEL. But in the SQL statement in Figure 90, VGBEL is a column on VBRP, and adding indexes to document tables is not often done. (Though it may be done on occasion.)

Just for completeness, check the cardinality of VGBEL using ST04 DB2 catalog browser, to see if it might be a candidate for a new index.

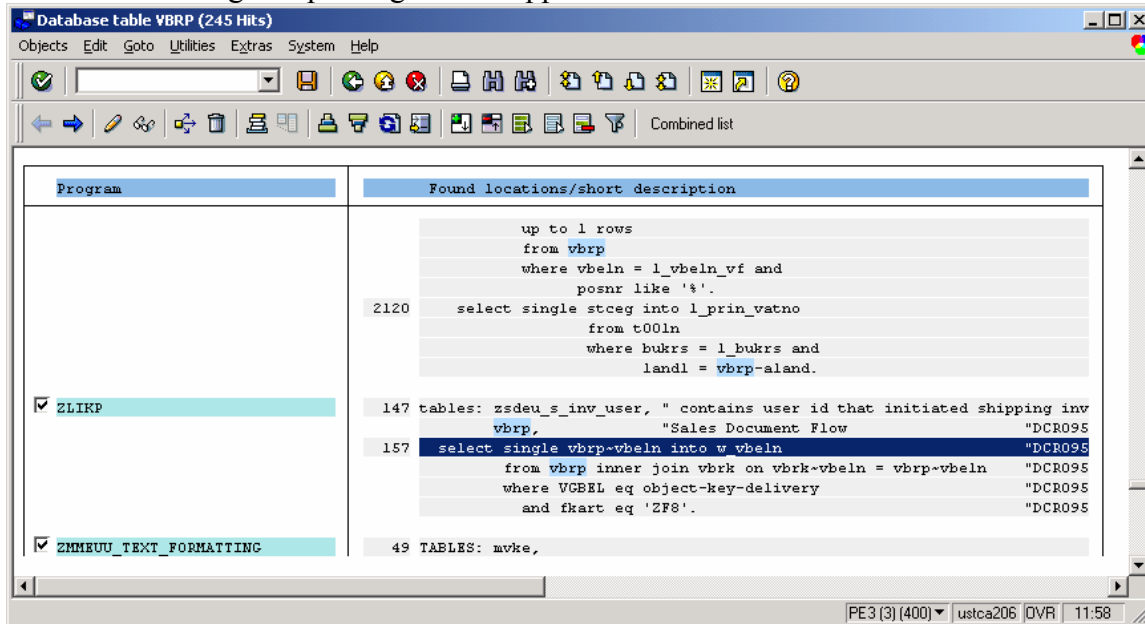


Cardinality of VGBEL is high (39K), it would filter the rows well, if it were indexed.





Now, look for where the statement comes from. Use SE11 “where used” on VBRP. Select programs in the “used in” popup. Select the Z\* programs, then press “detail view lines”, which is the icon with the green plus sign in the upper left.



The statement is in ZLIKP, a custom program (Z\*), not in the SAP namespace. This makes us suspicious that the ABAP may not be coded in the most efficient way. Note that “select single” is used to read the row, though this should be an ABAP “select”, since it is not fully qualified primary key access. STAT would report this as long “direct read” time.

In the code, one can see that the program is using a delivery number (object-key-delivery) to locate a billing document. But VBFA also contains these relationships, and it is indexed for searches based on predecessor (e.g. order) and successor (e.g. delivery) documents.

Our course of action is to send this program back to development, to see if it can be changed using the document number known to the program (object-key-delivery) with VBFA and other sales tables to find the billing documents needed by the program.

#### 8.4.4. Impact of dynamic SQL on access path chosen by DB2

SAP uses dynamic SQL to execute SQL statements in ABAP programs. Statements are first prepared with parameter markers, which are placeholders for the execution-time parameters. Since parameter markers are used for prepare, DB2 cannot make use of all the optimizer capabilities. For example, the column distribution statistics gathered with RUNSTATS FREQVAL (the TYPE=F statistics in SYSIBM.SYSCOLDIST), which track the values that most often occur in a column, are not used when optimizing dynamic SQL prepared with parameter markers.

When the TYPE=F statistics are not used, it can cause situations where DB2 does not choose an index that would be the most efficient way to access the data. Generally, these are situations where the SQL has a predicate referencing an index with a low cardinality column (e.g. a column that is a flag for processing status, which might have only a few possible values). When the statement is optimized with parameter markers, DB2 does not know whether the statement is searching for the small percentage of unprocessed rows, or the large percentage of processed rows, and it chooses another way to access the data, rather than the low cardinality index. If the statement is re-optimized at execution, then DB2 will see the value of the predicate variable, and be able to compare it against the column distribution data, and can then choose the best access path by determining whether the value occurs frequently or rarely in the column.

This issue commonly happens on queue tables such as EDIDC, the SWW\* tables for workflow, BDCPV (ALE change pointers -- see below), etc.

For SAP systems running at 4.5B or later, the hint "USE VALUES FOR OPTIMIZATION" (SAPnote 162034) can be added to the ABAP to cause the statement to be re-optimized at execution, when the host variables are available. In this way, DB2 can take advantage of all the optimizer data available.

For SAP systems running a release before 4.5B, these situations can sometimes be addressed by making changes to the catalog statistics, in order to influence the optimizer. Changing the catalog statistics can have side-effects, and so should be used carefully, and only after careful review the problem and other statements that reference the table.

For sites running 4.5B or later, hints are the preferred way to solve this problem.

In this example, the candidate statement (SELECT MANDT, CPIDENT...) was second in total getpages on the system, though it is executed only once in this interval. The list was first sorted by getpages, then the “Timers” tab was selected. Infrequently executed statements are not usually a priority in performance tuning, but this was chosen as an example for this paper, as it shows the impact of optimization with parameter markers, rather than values.

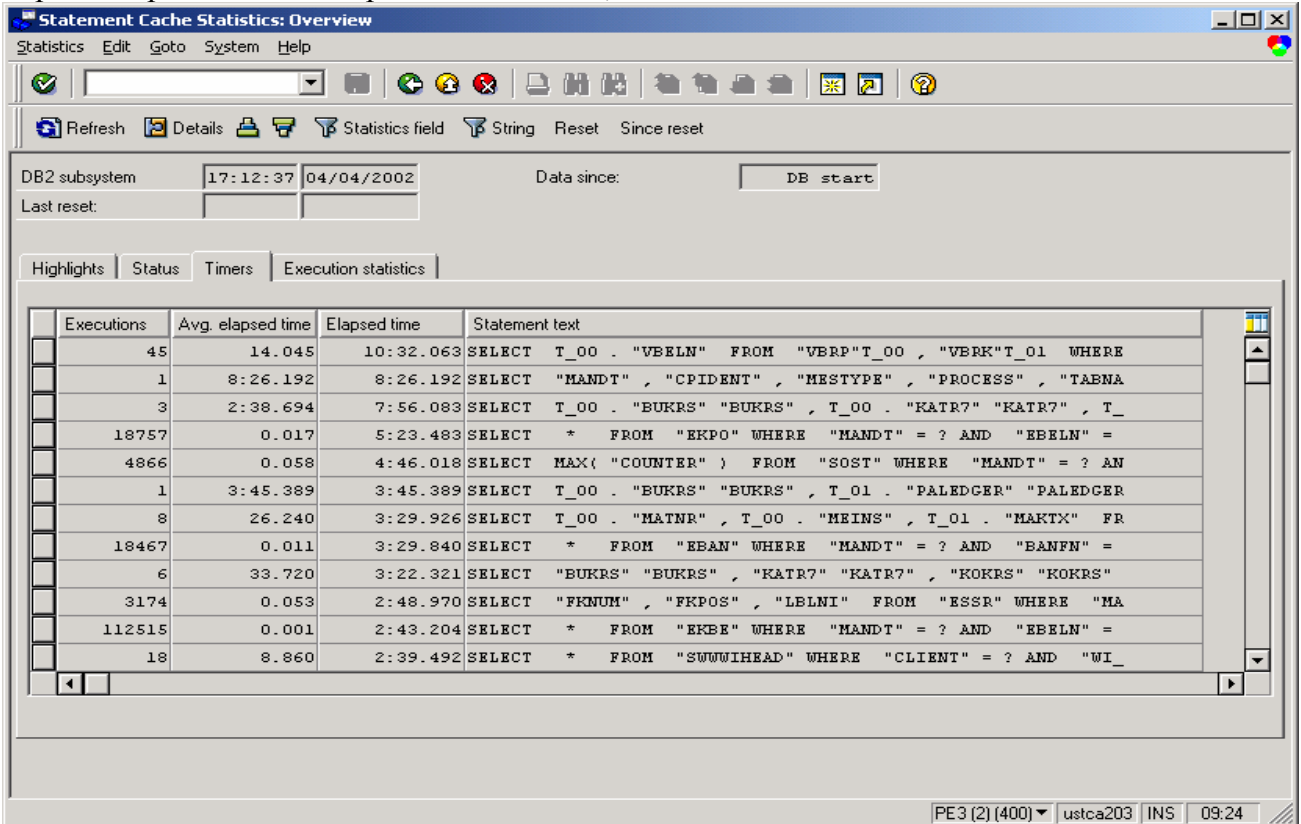


Figure 95: ST04 statement cache for BDCPV

Show the statement using the “details” button, and note the predicate columns – MANDT, MESTYPE, PROCESS, CETIME.

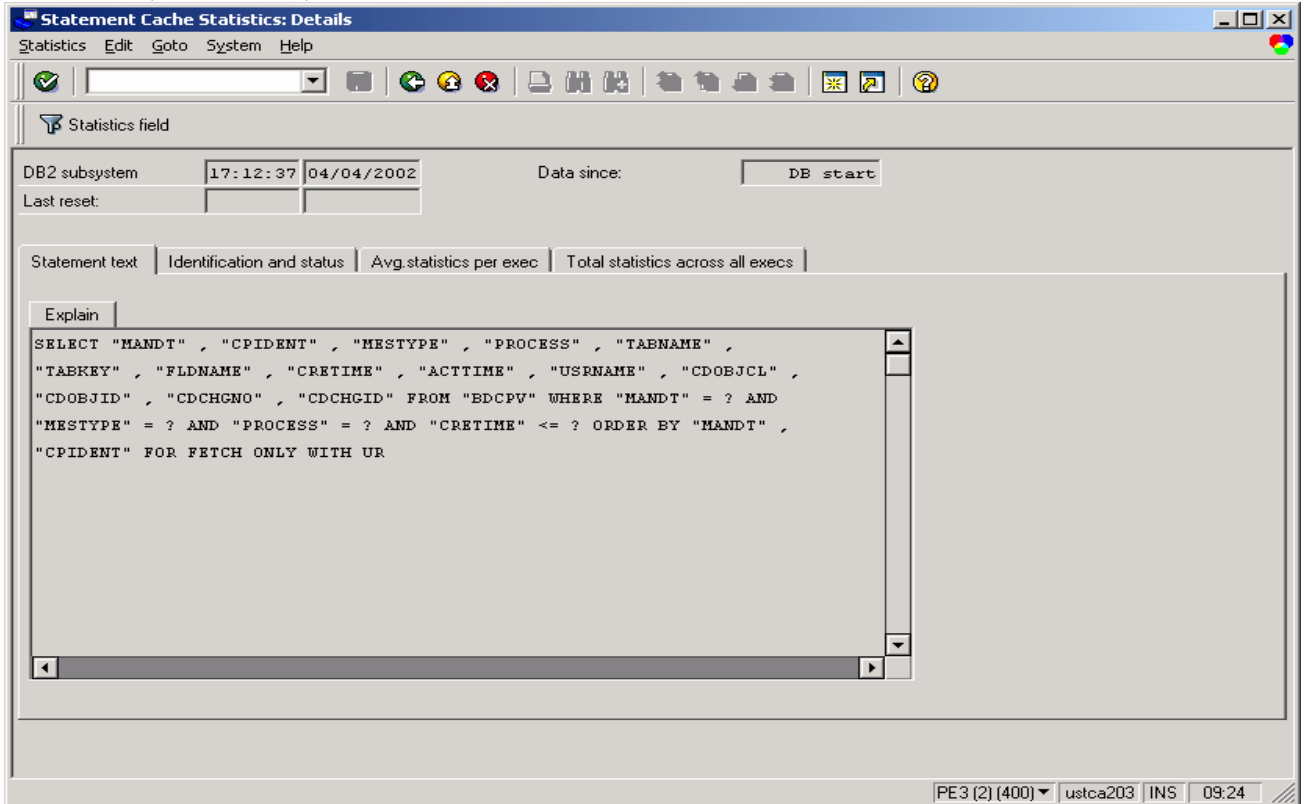


Figure 96: BDCPV statement

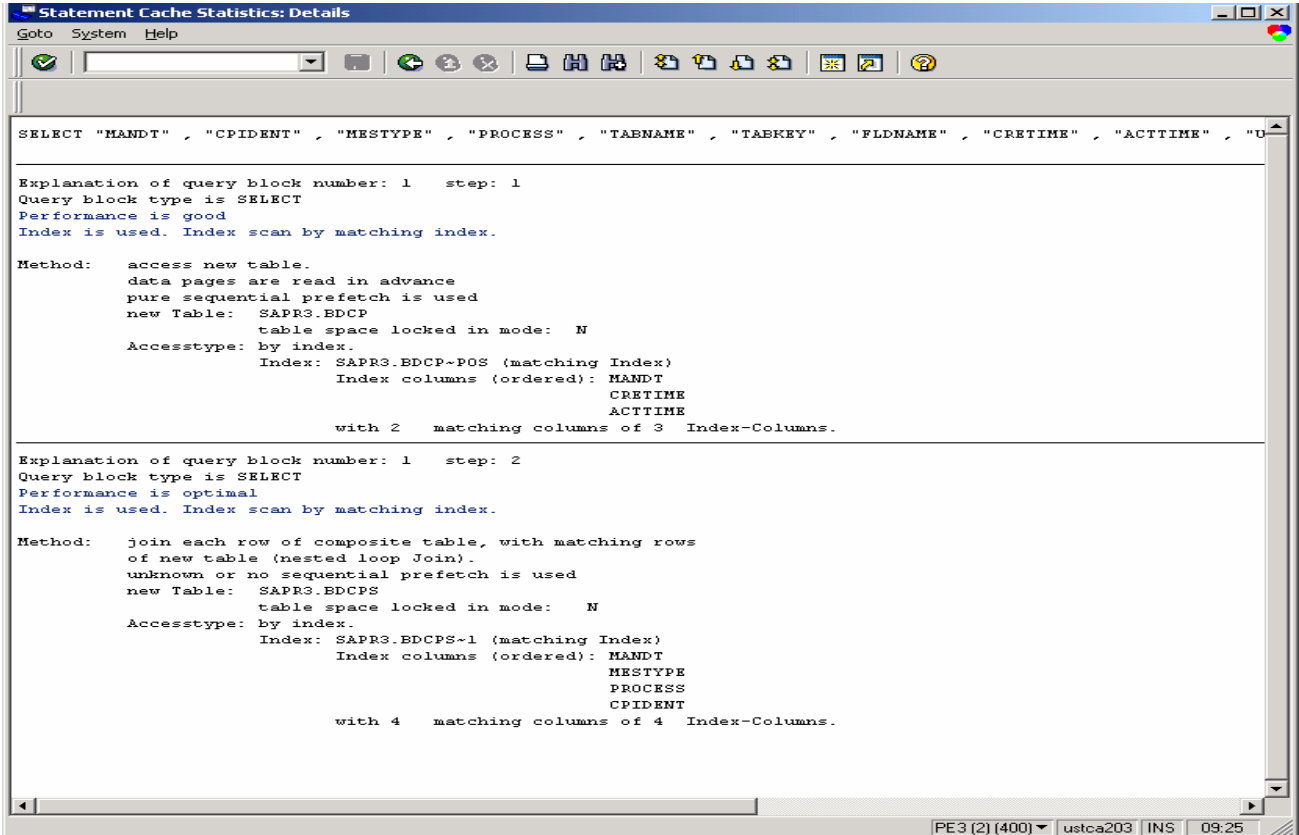
Check the “Avg. statements per exec” tab. We see that the statement does 736 getpages for every row processed. This is very inefficient. An efficiently indexed join might do about 10 getpages per row processed.

The screenshot shows the 'Statement Cache Statistics: Details' window. The 'Avg. statistics per exec' tab is selected, displaying a table of performance metrics. The 'Data since' field is set to 'DB start'.

Statement text	Identification and status	Avg. statistics per exec	Total statistics across all execs
Execs / sec		0.00	
Avg getpages		5761106.00	
Rows exam/execs		6373255.00	
Rows proc/execs		7820.00	
Getpages / processed		736.71	
Examined / processed		814.99	
Sync reads / execs		1891.00	
Synch writes / execs		0.00	
Avg sorts		1.00	
Avg idx scans		2116111.00	
Avg tbl scans		1.00	
Avg parallel groups		0.00	

Additional window details: DB2 subsystem, 17:12:37 04/04/2002, Last reset: [empty], Data since: DB start. Bottom status bar: PE3 (2) (400) | ustca203 | INS | 09:56

From the “statement text” tab in details, explain the statement. It is a view joining BDCP and BDCPS. Access to the inner table, BDCPS, uses all four columns of primary index, which should be efficient. This looks OK.



```

Statement Cache Statistics: Details
Goto System Help

SELECT "MANDT" , "CPIDENT" , "MESTYPE" , "PROCESS" , "TABNAME" , "TABKEY" , "FLDNAME" , "CRETIME" , "ACTTIME" , "U

Explanation of query block number: 1  step: 1
Query block type is SELECT
Performance is good
Index is used. Index scan by matching index.

Method:  access new table.
         data pages are read in advance
         pure sequential prefetch is used
new Table:  SAPRS.BDCP
         table space locked in mode:  N
Accessstype:  by index.
         Index:  SAPRS.BDCP-POS (matching Index)
         Index columns (ordered):  MANDT
                                   CRETIME
                                   ACTTIME
         with 2  matching columns of 3  Index-Columns.

Explanation of query block number: 1  step: 2
Query block type is SELECT
Performance is optimal
Index is used. Index scan by matching index.

Method:  join each row of composite table, with matching rows
         of new table (nested loop Join).
         unknown or no sequential prefetch is used
new Table:  SAPRS.BDCPS
         table space locked in mode:  N
Accessstype:  by index.
         Index:  SAPRS.BDCPS-1 (matching Index)
         Index columns (ordered):  MANDT
                                   MESTYPE
                                   PROCESS
                                   CPIDENT
         with 4  matching columns of 4  Index-Columns.

PE3 (2) (400) | ustca203 | INS | 09:25

```

Figure 97: BDCPV explain

Now, check the cardinality of the predicate columns using ST04 DB2 catalog browser.

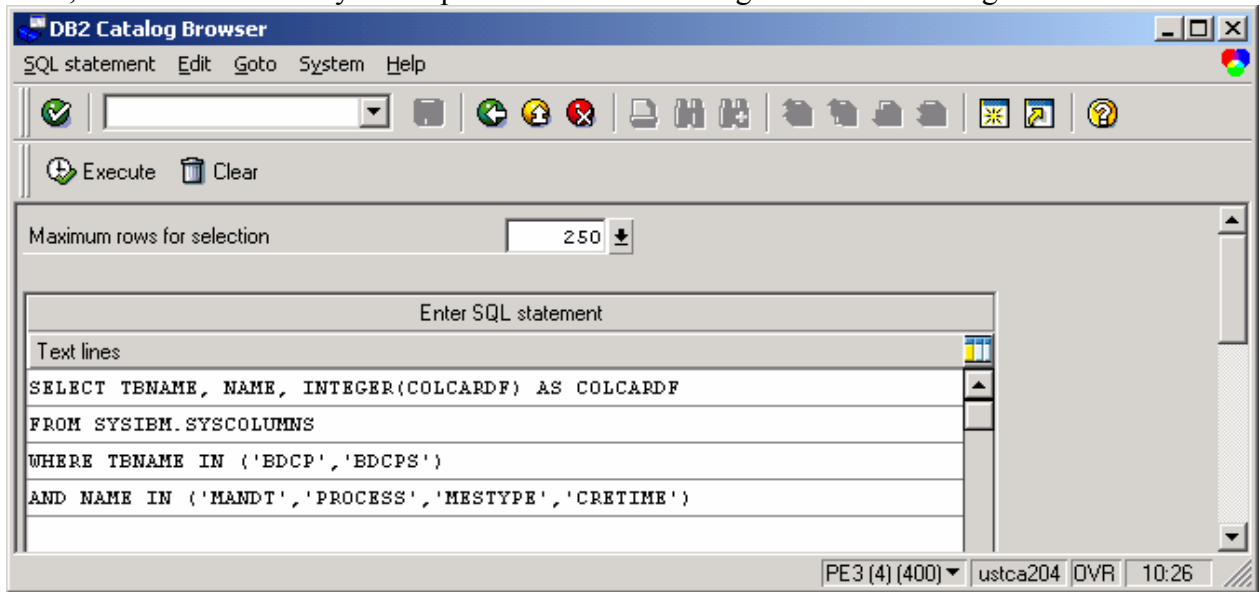


Figure 98: BDCPV query for cardinality of predicate columns

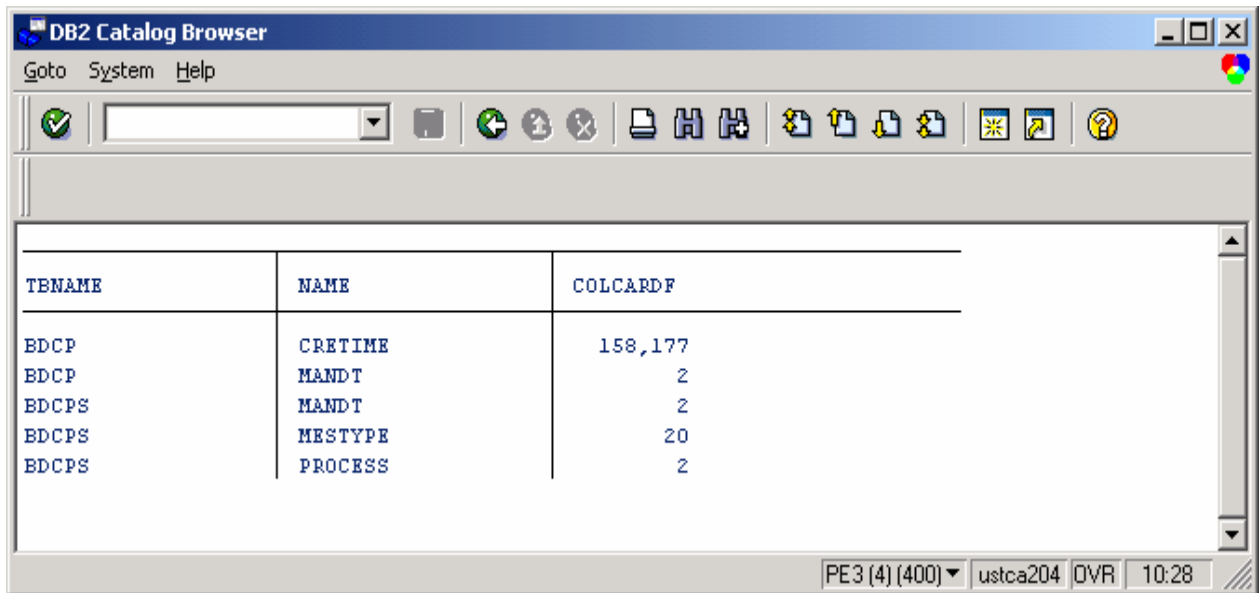


Figure 99: Column cardinality for BDCP and BDCPS

It looks reasonable – we see in Figure 99 that CETIME has the highest cardinality, and in explain plan above CETIME is in the index on the outer table. The SQL specifies CETIME >=, so we don't know how many rows might qualify. Thus, we are using an index with high cardinality on the outer table that should filter the rows well, and then accessing the inner table using all four columns of a four column index -- everything looks reasonable, but why is the statement so inefficient? Why does it still have to reference over 700 data pages to return one row? It must be that the candidate rows that pass the CETIME filter are being eliminated in the inner table (BDCPS), when MESTYPE and PROCESS are checked. (MANDT, MESTYPE, PROCESS, and CETIME are the predicates, and all are indexed in the join.) This seems perplexing, since MESTYPE and PROCESS have low cardinality (20 and 2) and look like they would not filter the rows.

Check ST04 DB2 catalog browser to see if there are other indexes that would match the predicates better.

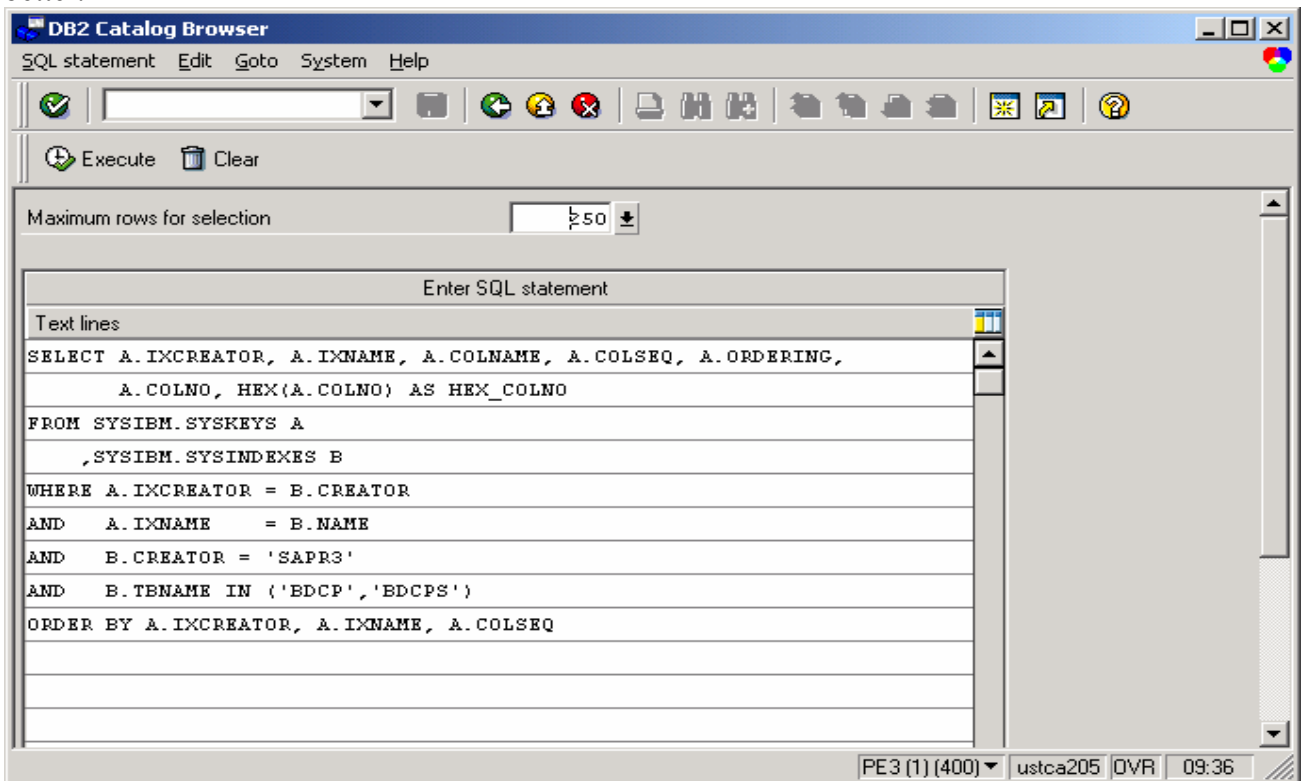


Figure 100: BDCPV query indexes on tables



There is an index (BSCPS~1) on PROCESS and MESTYPE. From the column cardinality information in Figure 99, one can assume that DB2 did not use this index with BDCPS for the outer table because of its low cardinality. If the column has low cardinality, DB2 thinks it will not filter the result set well. That is, DB2 thinks it will return lots of rows.

IXCREATOR	IXNAME	COLNAME	COLSEQ	ORDERING	COLNO	HEX_COLNO
SAPR3	BDCP~POS	MANDT	1	A	1	0001
SAPR3	BDCP~POS	CRETIME	2	A	6	0006
SAPR3	BDCP~POS	ACTTIME	3	A	7	0007
SAPR3	BDCP~Z01	TABNAME	1	A	3	0003
SAPR3	BDCP~Z01	CDOBJID	2	A	10	000A
SAPR3	BDCP~Z01	CDCHGID	3	A	12	000C
SAPR3	BDCP~0	MANDT	1	A	1	0001
SAPR3	BDCP~0	CPIDENT	2	A	2	0002
SAPR3	BDCP~1	CRETIME	1	A	6	0006
SAPR3	BDCP~1	CDOBJCL	2	A	9	0009
SAPR3	BDCPS~0	MANDT	1	A	1	0001
SAPR3	BDCPS~0	CPIDENT	2	A	2	0002
SAPR3	BDCPS~0	MESTYPE	3	A	3	0003
SAPR3	BDCPS~1	MANDT	1	A	1	0001
SAPR3	BDCPS~1	MESTYPE	2	A	3	0003
SAPR3	BDCPS~1	PROCESS	3	A	4	0004
SAPR3	BDCPS~1	CPIDENT	4	A	2	0002

Figure 101: BDCP and BDCPS indexes and columns

Look at BDCPS via SE11 in Figure 102, and see the issue. BDCPS is a status table and PROCESS is the flag (ALE processing indicator) that shows whether the row is done or not. PROCESS has cardinality 2 as we saw in Figure 99: “done” and “not done”. So, one can guess from the behavior of the SQL (700 getpages to return a row) that the statement is seeking a relatively small number of unprocessed rows, and that it would be better to have BDCPS as the outer table, using the existing BDCPS~1 index. This would filter out the vast majority of processed rows via an index on the outer table, rather than the inner table.

You can often see the list of possible values for a column such as PROCESS by using SE16, selecting the column in question, and pressing F4 to get a list of values.

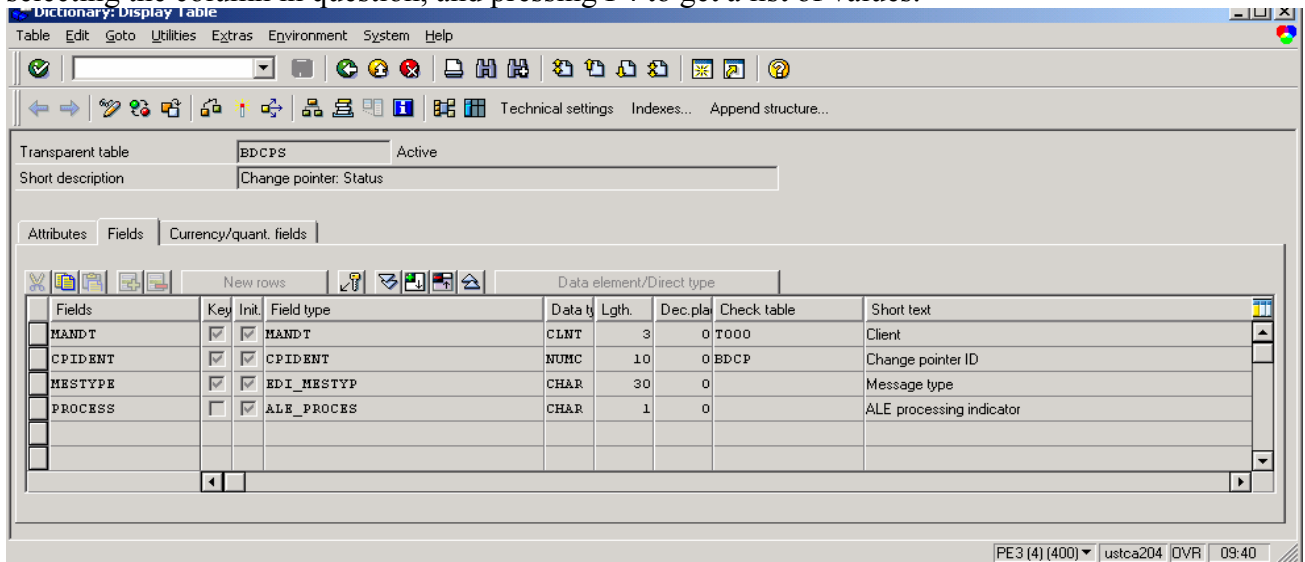
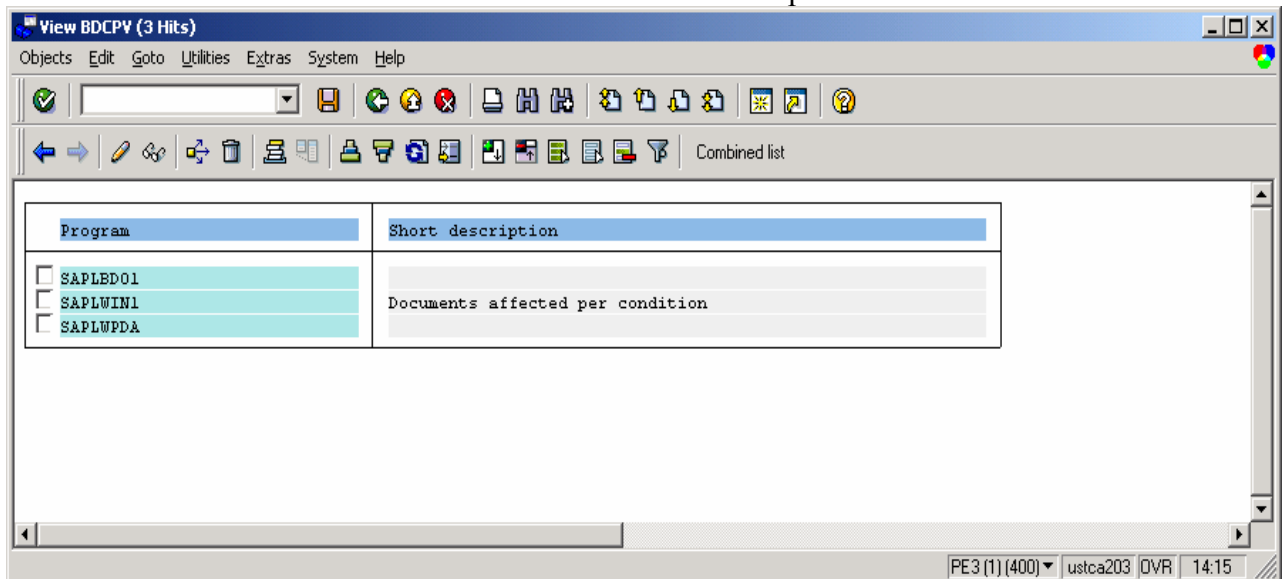


Figure 102: BDCPS table columns

Next do “where used” in SE11 on BDCPV to find where the problem comes from.

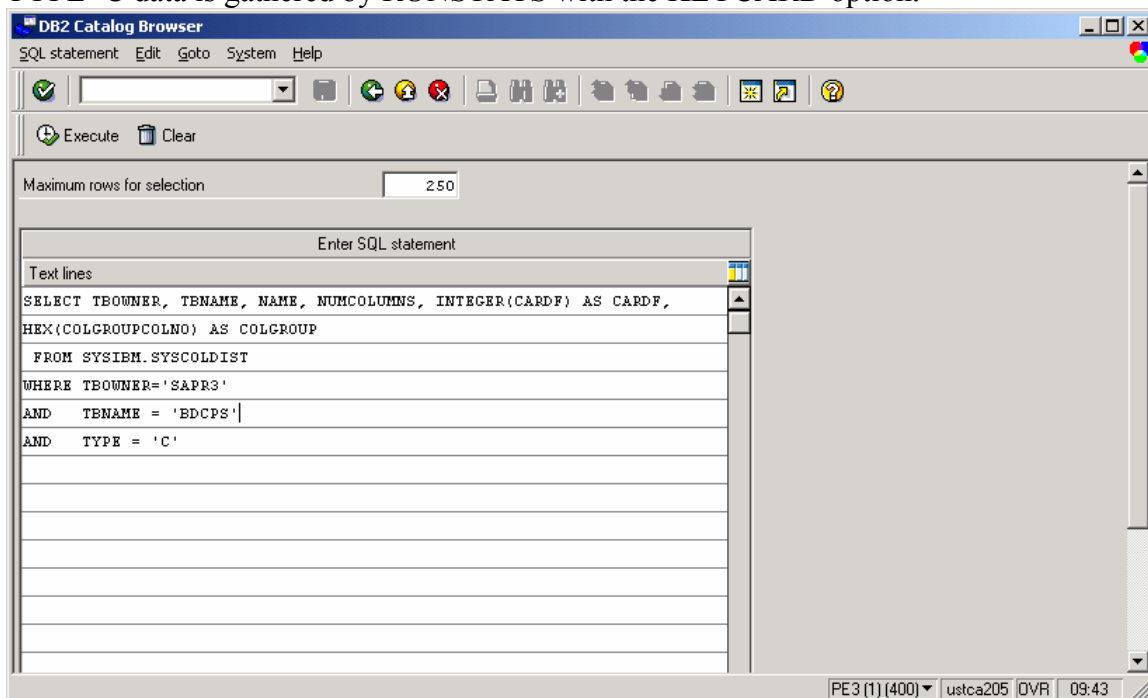


It is SAP code. If running SAP 4.5B or later, contact SAP regarding a code fix to add a hint to these programs, referring to SAPnote 162034. Then run RUNSTATS on these tables with the FREQVAL option, to gather the column distribution information that the optimizer needs in this case.

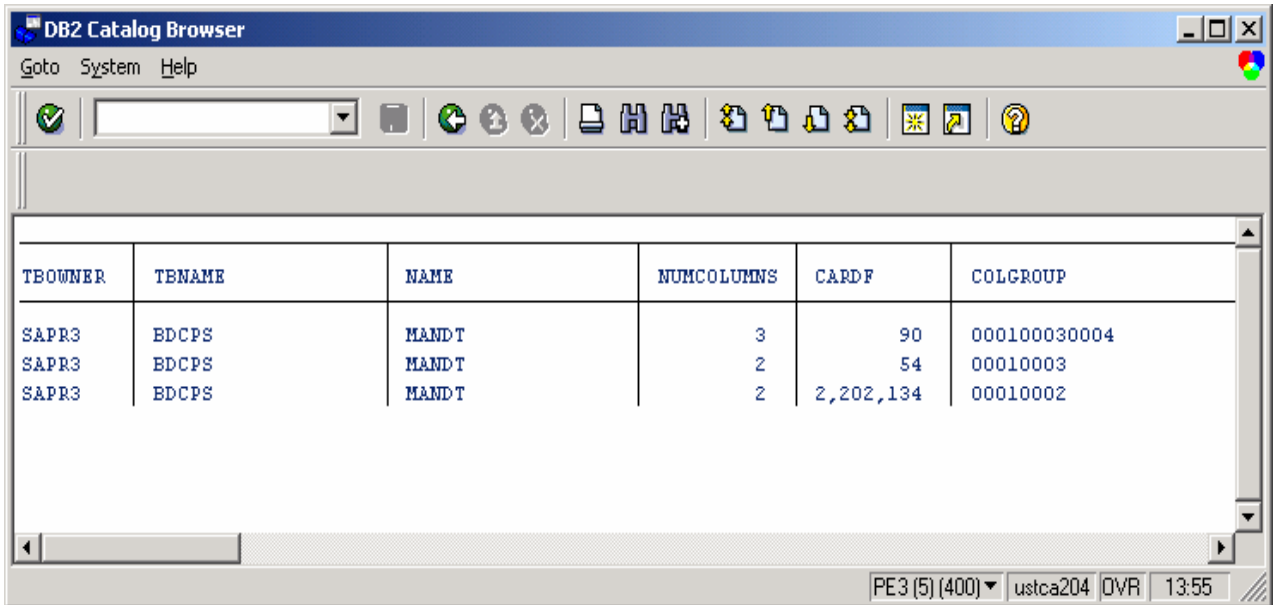
If running a version of SAP before 4.5B, then consider changing the catalog statistics. In order to do that, do some additional checks.

Check the predicate columns against the indexes on BDCPS, to find matches. See Figure 101, it would match the first three of the four columns (MANDT, MESTYPE, PROCESS, CPIDENT) in BDCPS~1.

Since the statement does not match all the columns in the index, check SYSCOLDIST for the cardinality of these three columns on the index, and get the name of the COLGROUP (concatenated columns from the left of index) for these three columns. The SYSCOLDIST TYPE=C data is gathered by RUNSTATS with the KEYCARD option.



**Figure 103: BDCPS KEYCARD statistics query**



TOWNER	TNAME	NAME	NUMCOLUMNS	CARDF	COLGROUP
SAPR3	BDCPS	MANDT	3	90	000100030004
SAPR3	BDCPS	MANDT	2	54	00010003
SAPR3	BDCPS	MANDT	2	2,202,134	00010002

**Figure 104: BDCPS KEYCARD statistics**

Check the HEX\_COLNO for MANDT, MESTYPE, and PROCESS in Figure 101. They are 0001, 0003, and 0004. So, COLGROUP 000100030004 corresponds to the three concatenated columns. In order to make BDCPS~1 a more preferred access path when only MANDT, MESTYPE, and PROCESS are specified in the SQL, increase CARDF on this COLGROUP. Make the following catalog changes to make COLGROUP 000100030004 more preferred by DB2.

Since there are about 6 million rows, and the select returned about 800 rows, set cardinality 10000. (6,000,000 rows / 10,000 distinct values = 600 rows average per distinct value.)

```
UPDATE SYSIBM.SYSCOLUMNS SET COLCARDF = 10000
WHERE TBCREATOR = 'SAPR3'
AND TNAME = 'BDCPS'
AND NAME = 'PROCESS'

UPDATE SYSIBM.SYSCOLDIST SET CARDF = 10000
WHERE TOWNER = 'SAPR3'
AND TNAME = 'BDCPS'
AND COLGROUP = X'000100030004'

UPDATE SYSIBM.SYSINDEXES SET FULLKEYCARDF = 10000
WHERE CREATOR = 'SAPR3'
AND TNAME = 'BDCPS'
AND NAME = 'BDCPS~1'
```

The ABAP hint “USE VALUES FOR OPTIMIZATION” causes DB2 to optimize using the variables, and then choose the best access path, based on detailed information DB2 has about the frequency with which values appear on columns. Contrast manual catalog statistics fixes with ABAP hints -- when we set the catalog statistics manually and do not use a hint, and a statement referencing BDCPS by MANDT, MESTYPE, and PROCESS is prepared, the catalog change will cause BDCPS~1 to be chosen by the DB2 optimizer – when “USE VALUES FOR

OPTIMIZATION” is in the ABAP, DB2 will check whether infrequently occurring values are being selected, before using the index.

Manually setting the catalog statistics could cause some programs to run slower. An example would be when a program wants the processed rows that make up most of the table. If a program wants 99% of rows in the table being processed, a tablescan would be the best choice. Therefore, one must evaluate how large the impact of the problem is, and what benefit would be expected, before making changes to catalog statistics tables. One must also check other statements that access the table, to ensure that the statistics change does not affect the wrong statements.

### 8.4.5. I/O constraint on table

While I/O constraints can cause average I/O times displayed in ST04 “times” to increase, the constraint will generally be seen most clearly on statements that access datasets on the overloaded volumes.

This example is from a 4.0B SAP system, so the format of ST04 is different than the previous examples. Follow the usual process for filtering ST04 statements - sort ST04 statement cache by elapsed time, and examine statements on the top of the list. The “SELECT \* FROM PROP” statement is at the top in elapsed time.

Though the ST04 format is old, and less clear than the new format, this example was chosen to show how to drill down from SAP into OS/390 programs to find an I/O constraint. With the advent of ESS disk, with its large cache and PAV capabilities, volume-level I/O constraints are now less common than they were.

The average elapsed time (320 ms per single row select) is extremely long. A well-indexed statement that returns a single row usually takes a few ms at the most.

Executions	Rows examined	Rows processed	Accumulated elap. time	Rows exam. / execs	Rows proc. / execs	Rows proc. / examined	Elap. time / execs	Statement text
1072923	2145746	1072836	003 23:24	2.000	1.000	0.500	320.151	SELECT * FROM "PROP"
2802	4024190	2011358	003 19:10	1436.185	717.829	0.500	117141.835	SELECT * FROM "PROP"
1072844	1072899	1072819	12:08:11.178505	1.000	1.000	1.000	40.724	UPDATE "PROP" SET "PNUM2
1654877	10226182	5113045	9:03:57.601127	6.179	3.090	0.500	19.722	DELETE FROM "PROP" WHERE
1966606	6599038	5211246	5:53:15.053725	3.356	2.650	0.790	10.777	SELECT * FROM "MVER"

**Figure 105: ST04 cached statement statistics with long select times on PROP**

The statement has low rows examined to rows processed so the problem does not look like access path. The indicator is inverted in this 4.0 system – rows proc/examined = 0.5, which is 2 examined per processed. Press the “details two” button to go to screen two of the statement cache statistics, to check if the problem is index screening, since index screening will show a good ratio of rows examined to rows processed, but bad ratio of getpages to rows processed.



```

-----
|Synchronous | Getpage | Sort | Index | Tablespace |Buffer write|Statement
|buffer reads| operations | operations | scans | scans | operations |text
-----
| 1530440 | 4837174 | 0 | 1072890 | 0 | 0 |SELECT * FROM "PROP"
| 938483 | 6720696 | 0 | 2803 | 0 | 0 |SELECT * FROM "PROP"
| 21792 | 4945110 | 0 | 1072689 | 0 | 0 |UPDATE "PROP" SET "PNUM2
| 1852470 | 53957375 | 0 | 3308476 | 0 | 0 |DELETE FROM "PROP" WHERE
| 1222198 | 2456890 | 0 | 1966545 | 0 | 0 |SELECT * FROM "MVER"
-----

```

Figure 106: ST04 statement cache “details two” with PROP long elapsed time

The statement executed 487174 getpages in 1072923 executions (see first screen), for about 5 getpages per execution. This is fine. It could not possibly take 300 ms to do 5 getpages unless there is some other problem.

Next, drill down to the OS/390 level, and check for device delays using RMF III DEV command. In this example, we’re looking at the system while it is running. If we needed to use historical statistics, we could use RMF III for recent history, or RMF I DASD and CACHE(SUMMARY) reports to look for volumes that are active and have long response times.

```

RMF 2.4.0 Device Delays                               Line 1 of 9

Samples: 299      System: SAP2 Date: 01/15/00 Time: 19.00.00 Range: 300 Sec

          DLY USC CON  ----- Main Delay Volume(s) -----
Jobname  C DMN  PG   %   %   %   % VOLSER  % VOLSER  % VOLSER  % VOLSER
-----
SB1DBM1  S   7  21  100 100 485 100 SB1198  70 SB1036  27 SB1147  16 SB1534
QASDBM1  S   7  31  17  19  20   8 QAS458   5 QAS009   1 QAS016   0 QAS266
DEVDBM1  S   7  27  16   8   8  15 DEV008   1 DEV015   1 DEV021   0 DEV004
SB1MSTR  S   7  22   3  58  97   1 SB1002   1 SB1006   1 PRJ021   0 SB1164
DFSMHSM  S   6   4   3  11   9   2 SYS291   0 PRJ001   0 PRJ051   0 SYS200
CATALOG  S   0   0   2   6   4   1 QASCAT   0 PRJ100   0 SB1CAT   0 JA3CAT
*MASTER* S   0   0   2   3   3   1 PAG201   1 PAG202   0 QAS383   0 QAS057
DEVMSTR  S   7  28   1   1   1   1 DEV026
VVICTOR  T  ***  ***   1   0   0   0 SPJ200   0 JA3RS1

```

Figure 107: RMF III DEV report for 300 second interval

Looking at volume delays, SB1198 and SB1036 are the two tops for the DBM1 address space. Note that RMF III reports volume and dataset delays against DBM1, even though threads issuing synchronous I/O requests are created under the ICLI address space.

In contrast, RMF I reports asynchronous I/O under DBM1, synchronous I/O under the ICLI, and logging I/O under the MSTR address space.



Look at the device activity and response times with RMF III “DEV R”. Note the 522 ms response time for SB1198.

```

RMF 2.4.0 Device Resource Delays                               Line 1 of 368

Samples: 299      System: SAP2  Date: 01/15/00  Time: 19.00.00  Range: 300  Sec

Volume S/  Act  Resp  ACT CON DSC PND %,  DEV/CU          USC DLY
 /Num  MX  Rate  Time  %  %  %  Reasons  Type          Jobname  C DMN PG  %  %

SB1198 S    144 .522   94  77  14 PND   3 33903  SB1DBM1 S   7  21  81 100
 1A71                               3990-3
SB1036 S    87.0 .064   68  51  15 PND   2 33903  SB1DBM1 S   7  21  58  70
 1090                               3990-3
SB1147 S    44.1 .020   57  54   2 PND   1 33903  SB1DBM1 S   7  21  57  27
 1230                               3990-3
SB1534 S    14.8 .025   18  15   3 PND   0 33903  SB1DBM1 S   7  21  20  16
 1166                               3990-3
SB1574 S    142 .010   24  21   1 PND   2 33903  SB1DBM1 S   7  21  26  15
 210E                               3990-3
DEV008 S    14.4 .023   17   8   9 PND   0 33903  DEVDBM1 S   7  27   8  15
 019A                               3990-3
    
```

Figure 108: RMF III DEV R report

Check what the active datasets are on the top two volumes with the RMF III “DSNV” command. On the most active volume, SB1198, check the “Data Set Name” field and see that the dataset is the PROP table. This is the same table that has the slow SQL statement that we saw in Figure 106.

```

RMF 2.4.0 Data Set Delays - Volume                               Line 1 of 1

Samples: 299      System: SAP2  Date: 01/15/00  Time: 19.00.00  Range: 300  Sec

----- Volume SB1198 Device Data -----
Number:    1A71      Active:    94%      Pending:    3%      Average Users
Device:    33903    Connect:   77%      Delay DB:   0%      Delayed
Shared:    Yes      Disconnect: 14%      Delay CU:   0%      1.0
                                           Delay DP:   0%
----- Data Set Name -----
SB1.DSNDBD.A120#JVY.PROP.I0001.A001  Jobname  ASID  DUSC%  DDLY%
                                           SB1DBM1 0083   81   100
    
```

Figure 109: RMF III DSNV report

Check the second most active volume, SB1036. The “Data Set Name” field shows a dataset for the “PROP~0” index. (Since the “~” used by SAP in index names is not valid in dataset names, it was converted to H by SAP when the dataset was created).

```

RMF 2.4.0 Data Set Delays - Volume                               Line 1 of 1

Samples: 299      System: SAP2  Date: 01/15/00  Time: 19.00.00  Range: 300  Sec

----- Volume SB1036 Device Data -----
Number:   1090      Active:    68%      Pending:    2%      Average Users
Device:   33903    Connect:   51%      Delay DB:   0%      Delayed
Shared:   Yes      Disconnect: 15%      Delay CU:   0%      0.7
                                           Delay DP:   0%
----- Data Set Name ----- Jobname ASID  DUSG% DDLY%
SB1.DSNDEB.A120#JVY.PROPHO.I0001.A001  SB1DBM1 0083  58   70

```

**Figure 110: RMF III DSNV report**

The RMF III command DSNJ, which is not pictured here, will give a summary of all active datasets for a job. Use DSNJ to display the datasets for the DBM1 address space when looking for I/O delays in an SAP system running DB2 for OS/390. (Though the threads under the ICLI initiate synchronous DB2 I/O, the I/O is not credited to the ICLI by RMF III. The RMF I WORKLOAD SUMMARY report does summarize synchronous I/O activity under the ICLI.)

Possible actions to resolve a volume level I/O constraint might be using ESS disk (since the PAVs give more concurrent I/Os per volume per second), or partitioning the table in DB2. PAVs can be used without change to the table structure. Partitioning requires analysis of the way that the table is accessed, to determine if there is a way to create partitions that will distribute the I/O activity.

### 8.4.6. Index screening

If an index has several columns (e.g. ONE, TWO, THREE, FOUR) and the SQL predicates referencing the columns contain a gap (e.g. ONE = AND TWO = and FOUR =), then DB2 will match the leftmost columns in the index, and may do index screening on the column after the gap. This is generally more efficient than examining the rows in the table to find the result, but can still result in inefficient SQL. Since the symptoms of index screening can be a little misleading, an example is included here.

Two of our three key indicators were getpages per row processed and examined per row processed. Here, note that examined per row looks very good (1.2), and getpages per row (2297) looks bad. If we were focusing only on examined per row, this would look like an efficient statement.



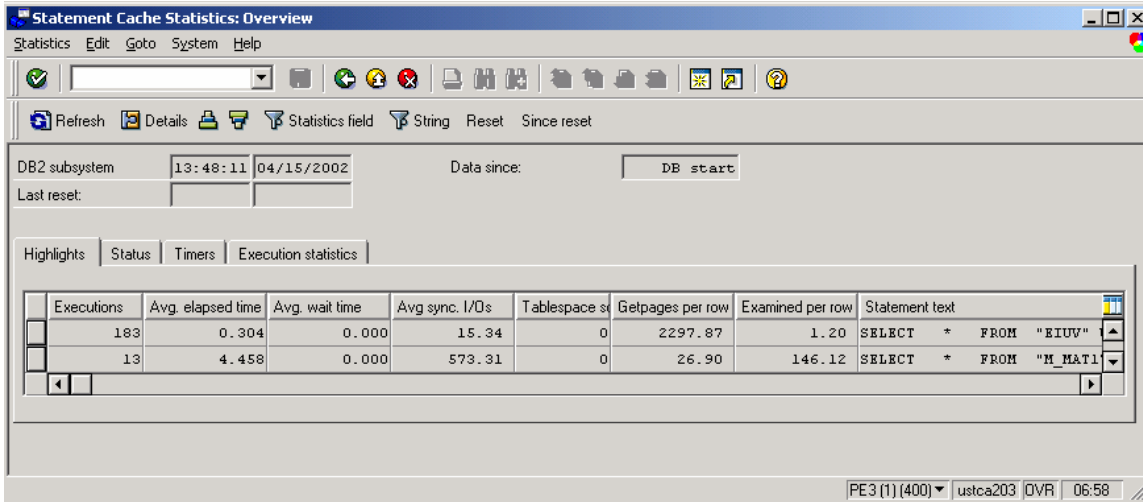


Figure 111: ST04 cached statement highlights with index screening

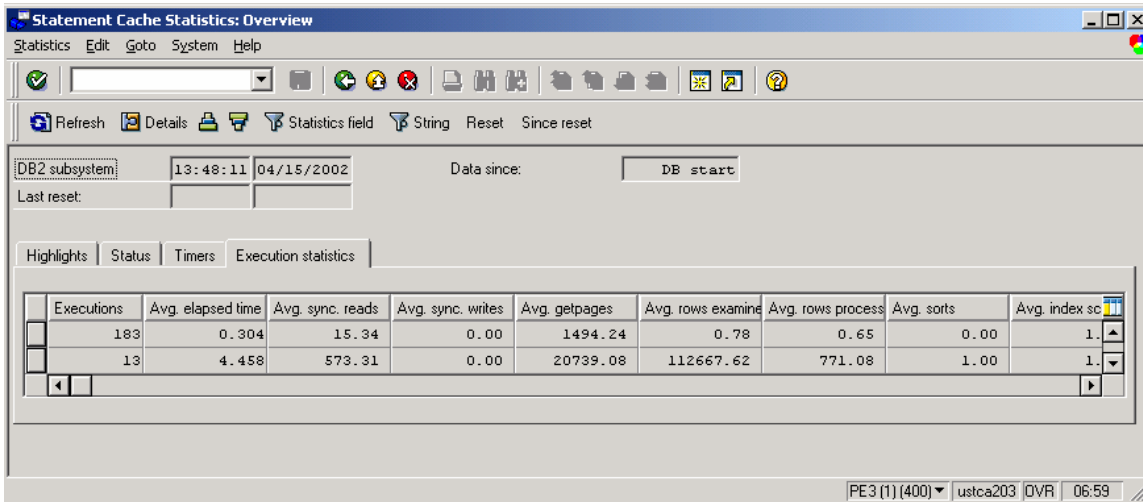


Figure 112: ST04 cached statement execution statistics with index screening

Next, display the statement via the details button, to check the predicates in the SQL.

The screenshot shows the 'Statement Cache Statistics: Details' window. The 'Statement text' tab is selected, displaying the following SQL query:

```
SELECT * FROM "BIUV" WHERE "HANDT" = ? AND "EXNUM" IN ( ? , ? , ? , ? , ? ,
? , ? ) FOR FETCH ONLY WITH UR
```

At the bottom of the window, the status bar shows 'PE3 (1) (400) | ustca203 | OVR | 07:00'.

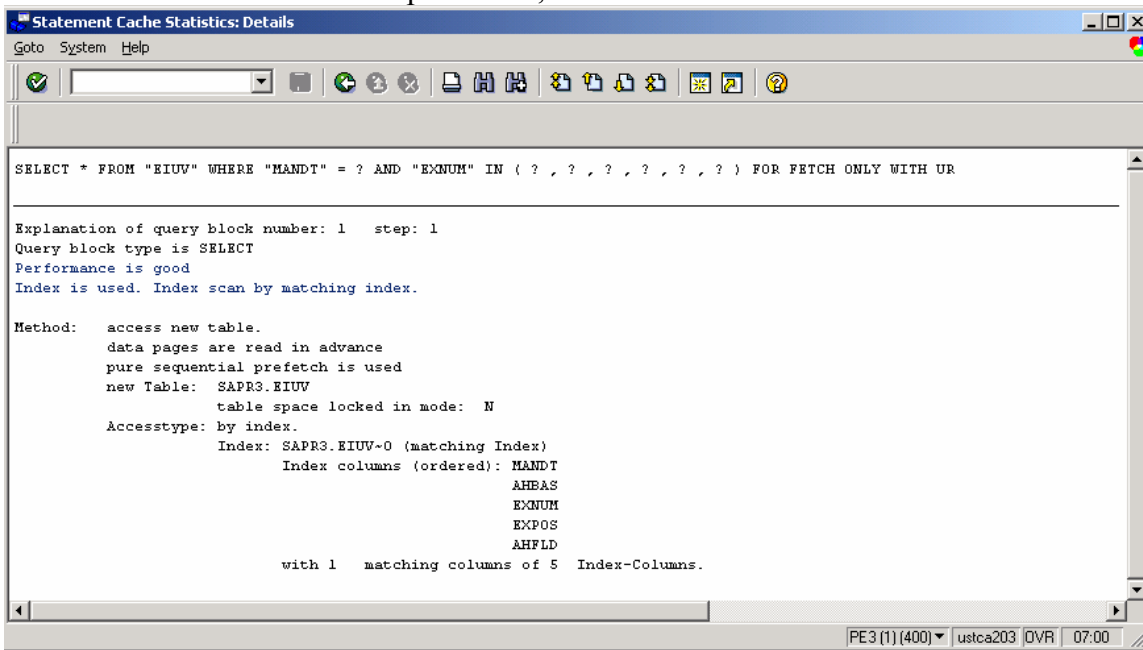
In execution statistics, see that it processes less than one row per execution.

The screenshot shows the 'Statement Cache Statistics: Details' window with the 'Avg. statistics per exec' tab selected. The following table displays the execution statistics:

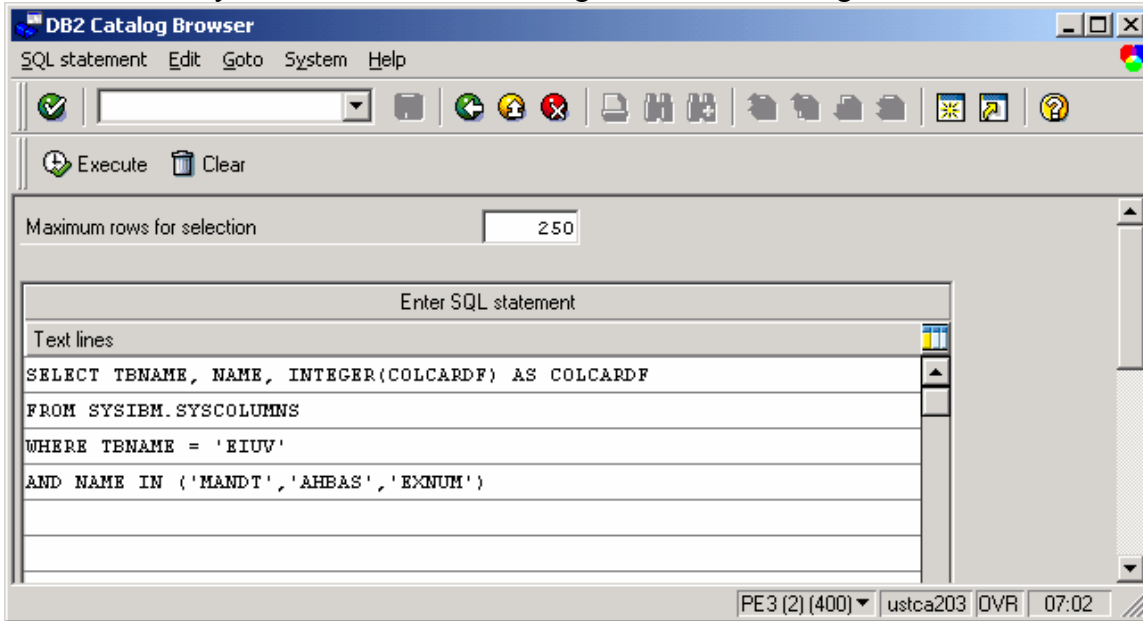
Statistic	Value
Execs / sec	0.00
Avg getpages	1494.24
Rows exam/execs	0.78
Rows proc/execs	0.65
Getpages / processed	2297.87
Examined / processed	1.20
Sync reads / execs	15.34
Sync writes / execs	0.00
Avg sorts	0.00
Avg idx scans	1.01
Avg tbl scans	0.00
Avg parallel groups	0.00

At the bottom of the window, the status bar shows 'PE3 (1) (400) | ustca203 | OVR | 07:04'.

In the “statement text” tab, explain shows that the statement matches only one column, MANDT, but if only one column of the index was used, and DB2 had to check all the rows in the table, there would not be such a large difference between getpages per row and examined per row. Note the EXNUM, a column in one of the statement predicates, is third column in the index.



Check cardinality of first three columns using ST04 DB02 catalog browser.



TBNAME	NAME	COLCARDF
EIUUV	AHBAS	4
EIUUV	EXNUM	17,051
EIUUV	MANDT	2

AHBAS will probably not help to filter the data, since it has low cardinality. If it were possible to include AHBAS in the SQL with an indexable predicate, then all three columns on the left of index 0 would match, and access would probably be much faster. Rather than searching all possible AHBAS values to ehcek EXNUM, DB2 could narrow the range to only one value of AHBAS.

Go looking for the source of the problem, with SE11 “where used”. Find the culprit, and note that the ABAP source does not look like the SQL statement. The ABAP “FOR ALL ENTRIES” construct uses rows in an ABAP internal table to select SQL. If one column is specified, then SAP will convert the statement to an “IN” list. If more than one column is specified, the statement will be converted to a UNION ALL.

Program	Found locations/short description
	1124 select * into table h_int_tab_eiuv from eiuv for all entries in h_int_tab_exnum where exnum eq h_int_tab_exnum-exnum.
LV50RFOG	116 FORM get_eikp_eipo_incompletness TABLES t_eiuv STRUCTURE eiuv USING s_eikp LIKE eikp s_eipo LIKE eipo v_level LIKE level v_glob_ahbas LIKE glob_ahbas v_incompl_fields TYPE c

Figure 113: FOR ALL ENTRIES

If this statement is executed many times when each program is run, then fixing the problem would offer a significant speedup for the program. If the statement is only executed a few times in each program, then fixing the problem would probably not be noticeable to end-users.

The actions to take in this case would be to first check whether the program could be changed to add an indexable predicate on AHBAS. If this were not possible, one could leave the program as it is and create an index (MANDT, EXNUM) on the table. Or, one could do nothing.

Before adding an index, the application and business impact of the problem would need to be analyzed, to determine how much value there is in speeding up the program.

### 8.4.7. Growing pains - catalog statistics out of date

At go-live, or when new functionality is added to an existing SAP system, it is very common for tables which were empty to grow quickly. At these times, RUNSTATS may need to be run more often than normal, to update the statistics of the tables as they start to be populated. After a couple weeks, the normal runstats cycle can generally be used.

Here is an example, from a 4.0B SAP system, of what can happen during this growth time. The statements in the cache were sorted by getpages, to bring the statements doing the most work to the top of the list. *Our candidate is in the second line*, the “SELECT \* from MDUP” statement.

Synchronous buffer reads	Getpage operations	Sort operations	Index scans	Tablespace scans	Buffer write operations	Statement text
733	87666511	0	0	0	0	DELETE FROM "MSTA" WHERE "MANDT" = ? AND "MATNR" =
426	55550651	0	311436449	0	0	SELECT * FROM "MDUP" WHERE "MANDT" = ? AND "MATN
1219	23172383	0	0	0	1	DELETE FROM "SNW_CONT" WHERE "CLIENT" = ? AND "WI_I
857	15874556	0	283420	0	0	SELECT "VBELN", "VGBEL", "VGPOS" FROM "LIPS" WHERE
1107593	15783511	0	14026256	0	0	SELECT "MANDANT", "OBJECTCLAS", "OBJECTID", "CHANGE
897	12649135	0	31000	0	0	SELECT * FROM "S910E" WHERE "MANDT" = ? AND "SSO
16835	9652795	0	152053	0	0	SELECT T_00 . "VBELN", T_00 . "VKORG", T_00 . "VDATU
72103	8998014	0	2530520	0	0	SELECT * FROM "MBEW" WHERE "MANDT" = ? AND "MATN
822	8815390	0	151208	0	0	SELECT T_00 . "VDATU", T_00 . "VBELN", T_00 . "VSBED
938	7737990	0	24652	0	0	SELECT "ERDAT", "ERZET", "CD_STATUS" FROM "DIK29"
286	7652368	0	283420	0	0	SELECT T_01 . "VBELN", T_01 . "POSNR", T_00 . "KNUMV

Figure 114: MDUP statement in ST04

Look at the “Details 1” screen for per-statement statistics.

Executions	Rows examined	Rows processed	Accumulated elap. time	Rows exam. / execs	Rows proc. / execs	Rows proc. / examined	Elap. time / execs	Statement text
578	19051382	0	32:26.411445	32960.869	0.000	0.000	3367.494	DELETE FROM "
63463	486311445	974	4:07:51.329053	7662.913	0.015	0.000	234.331	SELECT * FR
64646	4849394	0	8:12.737980	75.015	0.000	0.000	7.622	DELETE FROM "
28342	34580	17290	20:07.242035	1.220	0.610	0.500	42.596	SELECT "VBELN"
14026425	28052560	14026280	1:54:24.224185	2.000	1.000	0.500	0.489	SELECT "MANDAN
31000	93575	62005	13:41.781390	3.019	2.000	0.663	26.509	SELECT * FR
592	41159787	40275	13:20.436486	69526.667	68.032	0.001	1352.088	SELECT T_00 .
2530532	5059834	2529916	14:23.294903	2.000	1.000	0.500	0.341	SELECT * FR
538	37479399	51780	10:41.521187	69664.310	96.245	0.001	1192.419	SELECT T_00 .
24652	20248840	2410	3:37.073075	821.387	0.098	0.000	8.806	SELECT "ERDAT"
28342	0	0	10:33.121567	0.000	0.000	0.000	22.338	SELECT T_01 .

Figure 115: MDUP per statement statistics in ST04

Each time the statement is executed, it examines 7662 rows, and returns less than one row. Each execution takes 234 ms, which is very slow for a single row.

Drill into the statement, to see the full statement text.

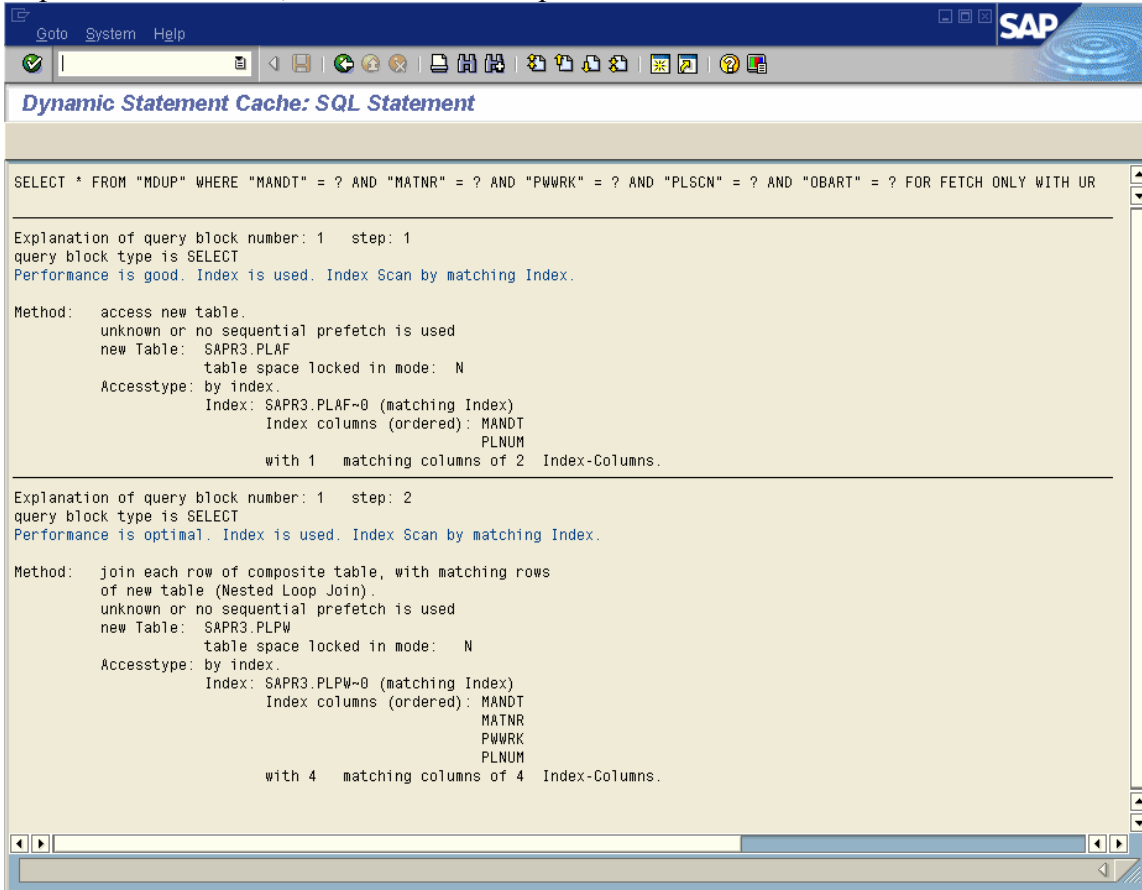
DB2 Statement

```

SELECT * FROM "MDUP" WHERE "MANDT" = ? AND "MATNR" = ? AND "PWRK" = ?
AND "PLSCN" = ? AND "OBART" = ? FOR FETCH ONLY WITH UR
    
```

Figure 116: MDUP statement

Explain the statement, to see what access path is used.



**Figure 117: MDUP explain**

This does not look good. The MANDT column is the only column used on the outer table (PLAF), which means that for every row on the outer table that matches on MANDT, the PLAF table must be checked before the inner table (PLPW) row can be checked. All four columns on the inner table match, so at least the indexed access to PLPW is efficient.

Look at the predicates on the statement. Three of the columns in the PLPW~0 index are specified. So it might be better for DB2 to use the PLPW table as the outer table, get the PLNUM from it, then use the PLNUM value for matching MANDT and PLNUM on PLAF as the inner table. Why is DB2 choosing the order PLAF for outer table and PLPW for inner table instead?

Use DB02 to check the catalog statistics.

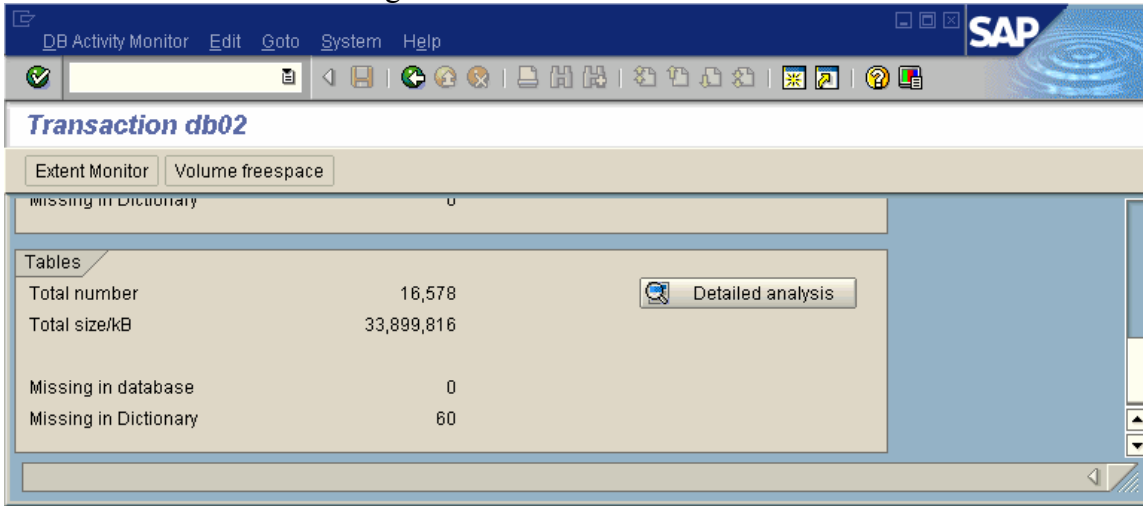
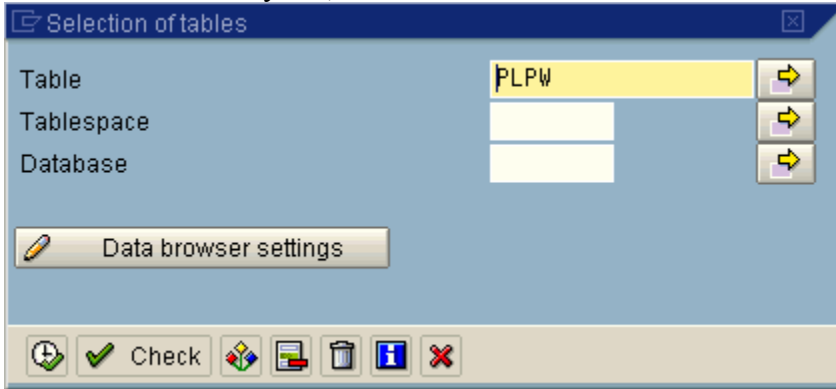


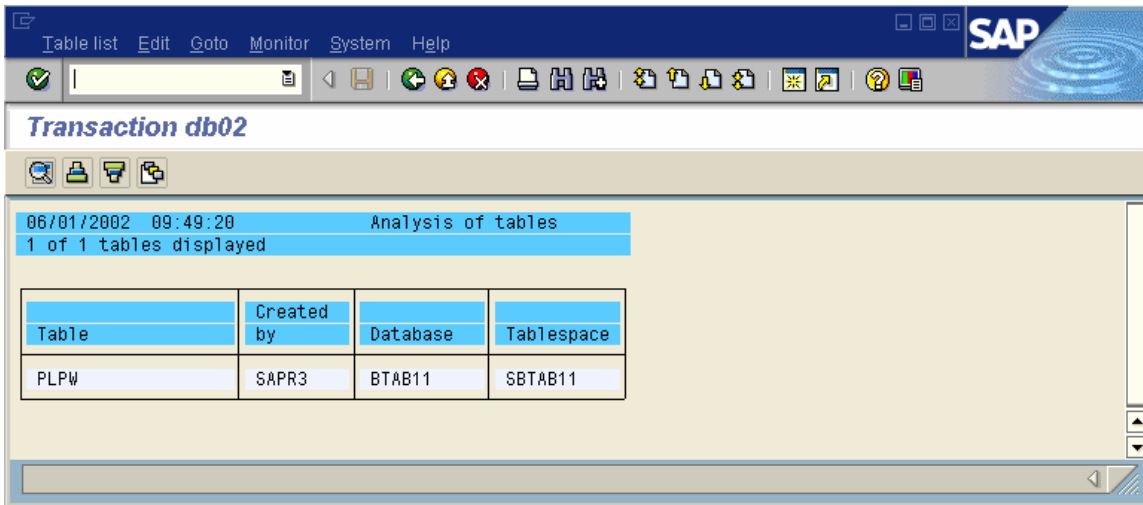
Figure 118: DB02 detailed analysis



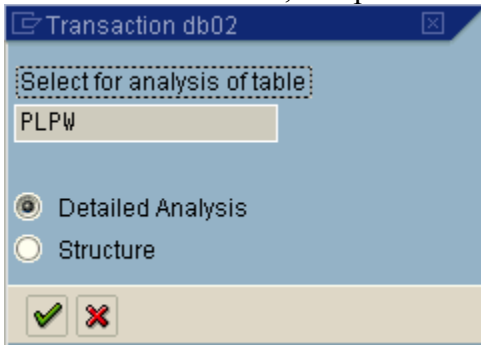
Press “Detailed analysis”, then enter the table name.



Press execute.



Then choose the table, and press the view icon (magnifying glass).



Select “Detailed analysis”, and press execute.

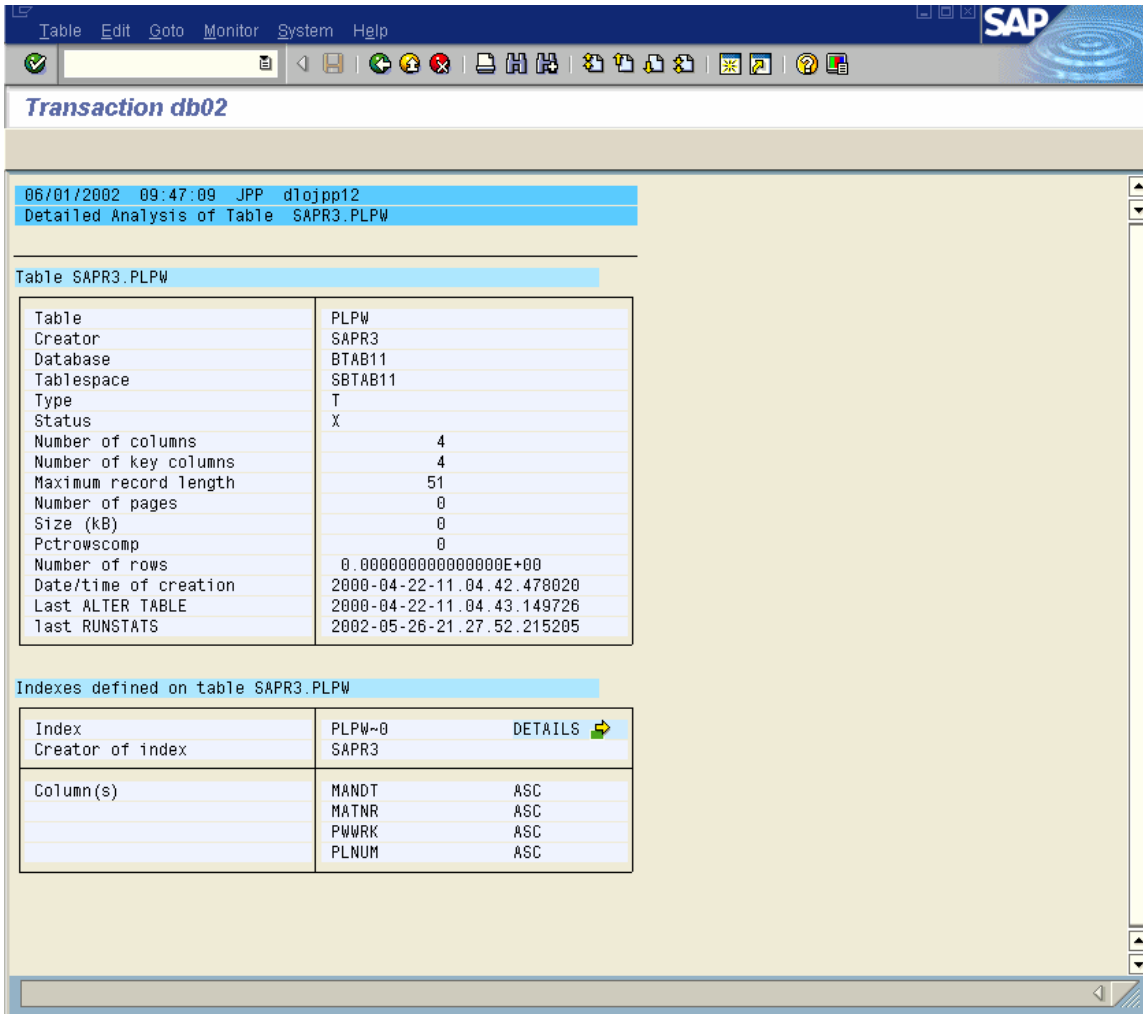


Figure 119: DB02 detail analysis to show catalog statistics

RUNSTATS was run on 5/26, but show no rows in the table. ST04 cache statistics showed that 7,000 rows are read on each execution, so it cannot be true that the table is empty.

Check PLAF in DB02 and the catalog statistics say it is also empty.

Transaction db02

06/01/2002 09:42:03 JPP dlojpp12  
Detailed Analysis of Table SAPR3.PLAF

Table SAPR3.PLAF

Table	PLAF
Creator	SAPR3
Database	BTAB11
Tablespace	SBTAB11
Type	T
Status	X
Number of columns	106
Number of key columns	2
Maximum record length	758
Number of pages	0
Size (kB)	0
Pctrowscomp	0
Number of rows	0.0000000000000000E+00
Date/time of creation	2001-09-08-11.55.30.829149
Last ALTER TABLE	2001-09-08-11.55.31.443079
Last RUNSTATS	2002-05-26-21.27.52.215205

Indexes defined on table SAPR3.PLAF

Index	PLAF~0	DETAILS
Creator of index	SAPR3	
Column(s)	MANDT	ASC
	PLNUM	ASC

Index	PLAF~1	DETAILS
Creator of index	SAPR3	
Column(s)	MANDT	ASC

Run RUNSTATS on the two tables and their indexes. After RUNSTATS, use the ST04 “cached statements” report, and filter the statements for MDUP table.

Our statement is the second line in the following MDUP display.

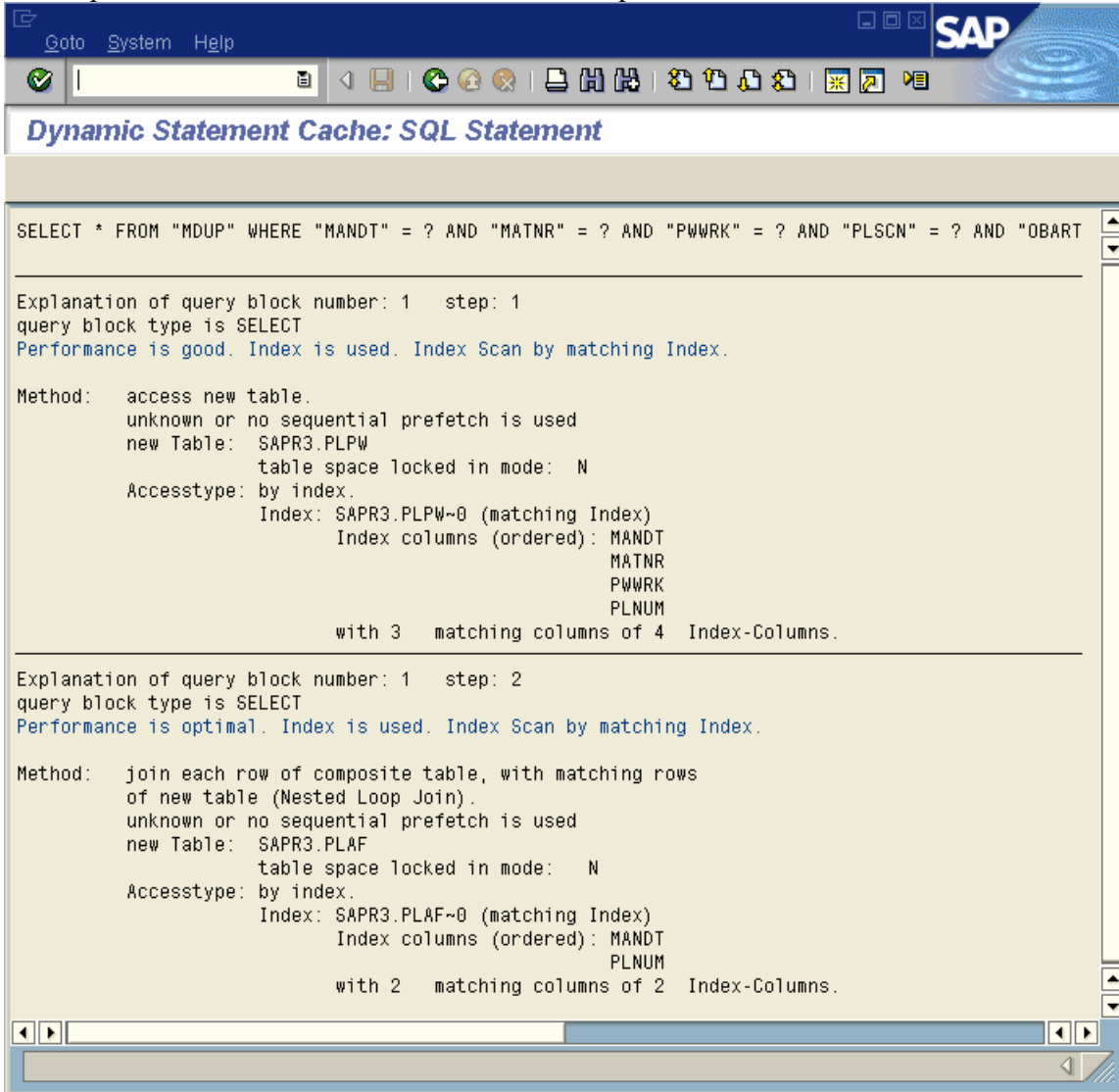
Prepared statement cache at 06/06/2002 04:49:03  
 Last reset: / / : : Data since: DB start

Executions	Rows examined	Rows processed	Accumulated elap. time	Rows exam. / execs	Rows proc. / execs	Rows proc. / examined	Elap. time / execs	Statement text
13762	86936	0	17.022564	6.317	0.000	0.000	1.237	SELECT * FROM "M
54962	108152	220	41.115259	1.968	0.004	0.002	0.748	SELECT * FROM "M

**Figure 120: MDUP statement statistics after runstats**

See the “Elap time / execs” column - the statement now takes less than one ms per execution.

Use explain to check the statement's new access path.



DB2 has changed the join order, and now takes PLPW as the outer table.

In this case, the RUNSTATS being out of date was obvious, since DB02 showed no rows in each table. One can have similar problems when RUNSTATS shows that a table is small, and it has suddenly grown.

If you suspect that the catalog statistics do not reflect the true size of the table, check the table row count (using SPUFI or SE16) and compare it against the catalog statistics.

### 8.4.8. Evaluating whether a new index is needed

Here is another example of checking whether the SQL can be improved by adding a new index. This is taken from a 3.1H system, so some calculations are required to interpret the statistics. Look at the *last statement* on the screen, “SELECT RCLNT...”. See that it examines over 10 rows for each row processed (4639 rows examined/348 rows processed), and takes over 100 ms (44592 ms/348 rows) per row processed. This looks like it can be improved. These are array operations (348 rows per execution) so we would expect the per-row times to be much better – less than one ms per row, in some cases.

Executions	Rows examined	Rows processed	Accumulated elap. time	Rows exam. / execs	Rows proc. / execs	Rows proc. / examined	Elap. time / execs	Statement text
3934721	44750532	2382155	002 22:29	11.670	0.621	0.053	66.175	SELECT * FROM
1182	2894	1581	9:10:31.768272	2.448	1.338	0.546	27945.658	SELECT "MANDT" , "
5969	2010102749	24480	2:48:52.521455	336757.036	4.101	0.000	1667.368	SELECT "VENUM" FR
51881	101511	52819	4:25:35.407385	1.957	1.018	0.520	307.153	SELECT "HUNNR" F
765905	8167101	0	1:09:19.776930	10.663	0.000	0.000	5.431	DELETE FROM "VEDAT
765902	6611958	0	48:06.132667	8.633	0.000	0.000	3.768	DELETE FROM "VBOD
3825298	36100643	32968995	7:02:36.850927	9.437	8.619	0.913	6.629	SELECT "MANDT" , "
7650850	13062350	4013454	18:07:21.568722	1.707	0.525	0.307	8.527	SELECT * FROM
95491	55553126	122502	34:18.704020	581.763	1.283	0.002	21.559	SELECT "MANDT" , "
3834711	27979403	9177108	3:02:54.088921	7.296	2.393	0.328	2.862	SELECT * FROM
6983	32397104	2436511	003 14:29	4639.425	348.920	0.075	44592.434	SELECT "RCLNT" , "

Figure 121: ST04 statement cache screen 1 – GLPCA

Synchronous buffer reads	Getpage operations	Sort operations	Index scans	Tablespace scans	Buffer write operations	Statement text
4286767	64739213	0	19372507	0	0	SELECT * FROM "MDUE" WHERE "MANDT" = ? AND "MATNR" =
549171	58110656	0	1182	0	0	SELECT "MANDT" , "TRSTA" , "VSTEL" , "TDDAT" , "VBELN" , "
142	54143613	0	0	5968	0	SELECT "VENUM" FROM "VEPO" WHERE "MANDT" = ? AND "VBEL
22702	53490690	0	51880	0	0	SELECT "HUNNR" FROM "KOPH" WHERE "MANDT" = ? AND "HUN
11720	49019032	0	0	0	0	DELETE FROM "VEDATA" WHERE "VBKEY" = ? *****
397	34655046	0	0	0	0	DELETE FROM "VBOD" WHERE "VBKEY" = ? *****
79010	33653289	0	1183036	0	0	SELECT "MANDT" , "BANFM" , "ENFPO" , "BSART" , "BSTYP" , "
881679	32819817	0	5033608	0	0	SELECT * FROM "MDBS" WHERE "MANDT" = ? AND "MATNR" =
35758	25625692	0	95492	0	0	SELECT "MANDT" , "EVT_ID" , "EVT_STATUS" , "OBJ_NAME" , "O
296346	25195506	0	13023567	0	0	SELECT * FROM "MDUA" WHERE "MANDT" = ? AND "MATNR" =
18159603	22763063	6983	6983	4902	0	SELECT "RCLNT" , "GL_SIRID" , "RLDNR" , "RRCTY" , "RVERS"

Figure 122: ST04 statement cache screen 2 - GLPCA

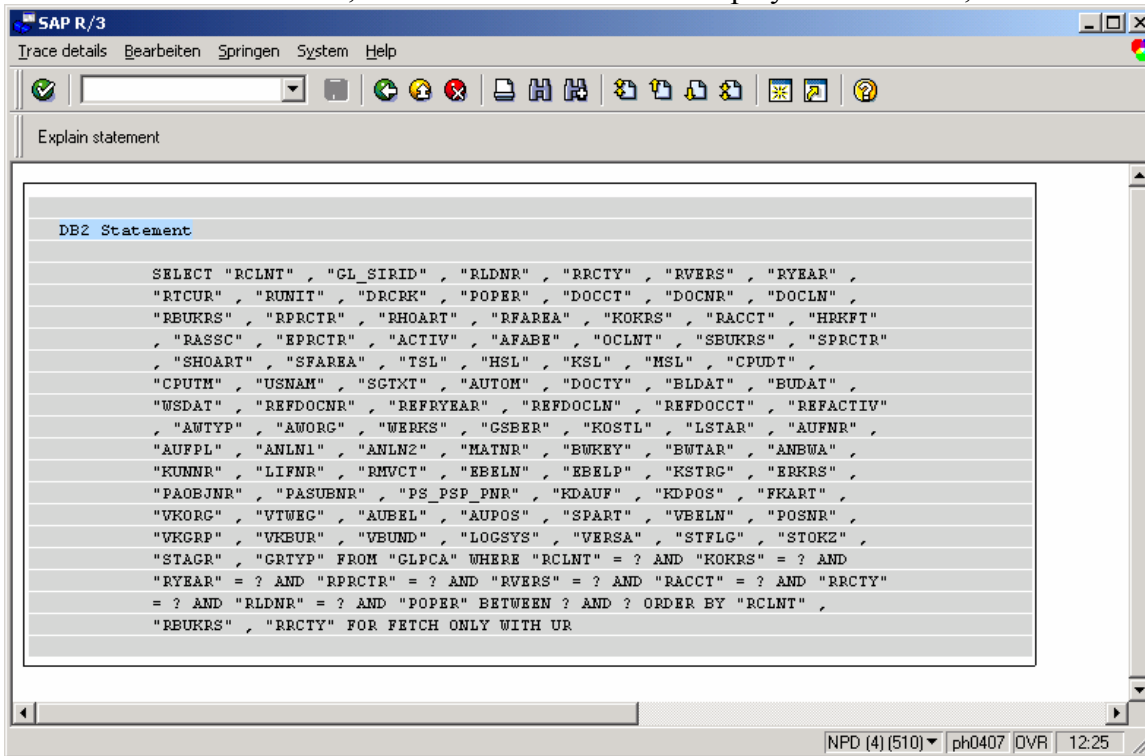
Combining the getpage information in Figure 122 with the execution statistics in Figure 121, the statement does about 9 getpages per row (22763063 getpages/2436511 rows), which is a bit high for single table indexed access. Perhaps we can find a way to get more efficient access to the table, reduce getpages per row, and improve the per-row response time.

While the program examines 10 rows for each row processed, and does 9 getpages for each row processed, it may not be not possible to optimize the statement to have one getpage per row processed. In general, with index processing and reading the data from the table, 3-4 getpages per row is good. Thus, one might hope for a 50-60% improvement in getpages per row here. In a few cases, such as

array fetch of many contiguous rows in a table, getpages per row may be very low, e.g. one getpage or less per row.

In Figure 122, note that about 50% getpages result in synchronous reads (22763063 getpages and 18159603 reads). The data needed is seldom in DB2 buffers, and must be read from disk. This is part of the reason that the per-row times are so long.

In ST04 cached statements, drill into the statement to display the statement, to determine the predicates.



The screenshot shows the SAP R/3 Trace details window with the 'Explain statement' tab selected. The main area displays the following DB2 statement:

```

DB2 Statement

SELECT "RCLNT" , "GL_SIRID" , "RLDNR" , "RRCTY" , "RVERS" , "RYEAR" ,
"RTCUR" , "RUNIT" , "DRCRK" , "POPER" , "DOCCT" , "DOCNR" , "DOCLN" ,
"RBUKRS" , "RPRCTR" , "RHOART" , "RFAREA" , "KOKRS" , "RACCT" , "HRKFT" ,
"RASSC" , "EPRCTR" , "ACTIV" , "AFABE" , "OCLNT" , "SBUKRS" , "SPRCTR" ,
"SHOART" , "SFAREA" , "TSL" , "HSL" , "KSL" , "MSL" , "CPUDT" ,
"CPUTH" , "USNAM" , "SGTXT" , "AUTOM" , "DOCTY" , "BLDAT" , "BUDAT" ,
"WSDAT" , "REFDOCNR" , "REFRYEAR" , "REFDOCLN" , "REFDOCCT" , "REFACTIV" ,
"AWTYP" , "ANORC" , "WERKS" , "CSBER" , "KOSTL" , "LSTAR" , "AUFNR" ,
"AUFPL" , "ANLN1" , "ANLN2" , "MATNR" , "BWKEY" , "BWTAR" , "ANBWA" ,
"KUNNR" , "LIFNR" , "RMVCT" , "EBELN" , "EBELP" , "KSTRG" , "ERKRS" ,
"PAOBJNR" , "PASUBNR" , "PS_PSP_PNR" , "KDAUF" , "KDPOS" , "FKART" ,
"VKORC" , "VTWEG" , "AUBEL" , "AUPOS" , "SPART" , "VBELN" , "POSNR" ,
"VKGRP" , "VKBUR" , "VBUND" , "LOCSYS" , "VERSA" , "STFLG" , "STOKZ" ,
"STAGR" , "GRTPY" FROM "GLPCA" WHERE "RCLNT" = ? AND "KOKRS" = ? AND
"RYEAR" = ? AND "RPRCTR" = ? AND "RVERS" = ? AND "RACCT" = ? AND "RRCTY"
= ? AND "RLDNR" = ? AND "POPER" BETWEEN ? AND ? ORDER BY "RCLNT" ,
"RBUKRS" , "RRCTY" FOR FETCH ONLY WITH UR

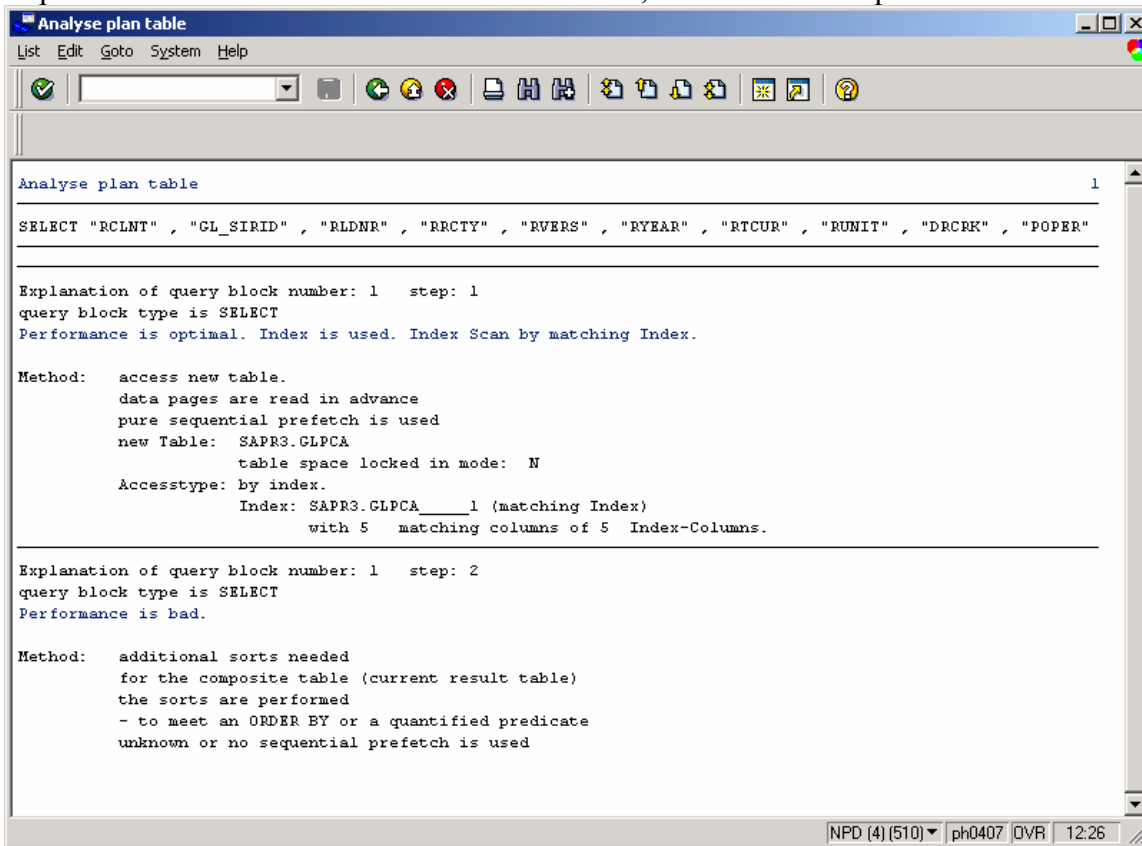
```

The status bar at the bottom of the window shows: NPD (4) (510) | ph0407 | OVR | 12:25

**Figure 123: GLPCA statement**

RCLNT, KOKRS, RYEAR, RPRCTR, RVERS, RACCT, RRCTY, RLDNR, and POPER are columns in predicates. POPER is a range predicate (BETWEEN), so DB2 cannot perform index processing on the columns on the right of POPER in an index. See the DB2 Administration guide, SC26-9003, for information on range predicates and how they affect SQL processing.

Explain the statement from ST04 statement cache, to see the access path used.



```

Analyse plan table 1
-----
SELECT "RCLNT" , "GL_SIRID" , "RLDNR" , "RRCTY" , "RVERS" , "RYEAR" , "RTCUR" , "RUNIT" , "DRCRK" , "POPER"
-----
Explanation of query block number: 1  step: 1
query block type is SELECT
Performance is optimal. Index is used. Index Scan by matching Index.

Method:  access new table.
         data pages are read in advance
         pure sequential prefetch is used
         new Table:  SAPR3.GLPCA
                table space locked in mode:  N
         Accesstype:  by index.
                Index:  SAPR3.GLPCA_____1 (matching Index)
                with 5  matching columns of 5  Index-Columns.
-----
Explanation of query block number: 1  step: 2
query block type is SELECT
Performance is bad.

Method:  additional sorts needed
         for the composite table (current result table)
         the sorts are performed
         - to meet an ORDER BY or a quantified predicate
         unknown or no sequential prefetch is used

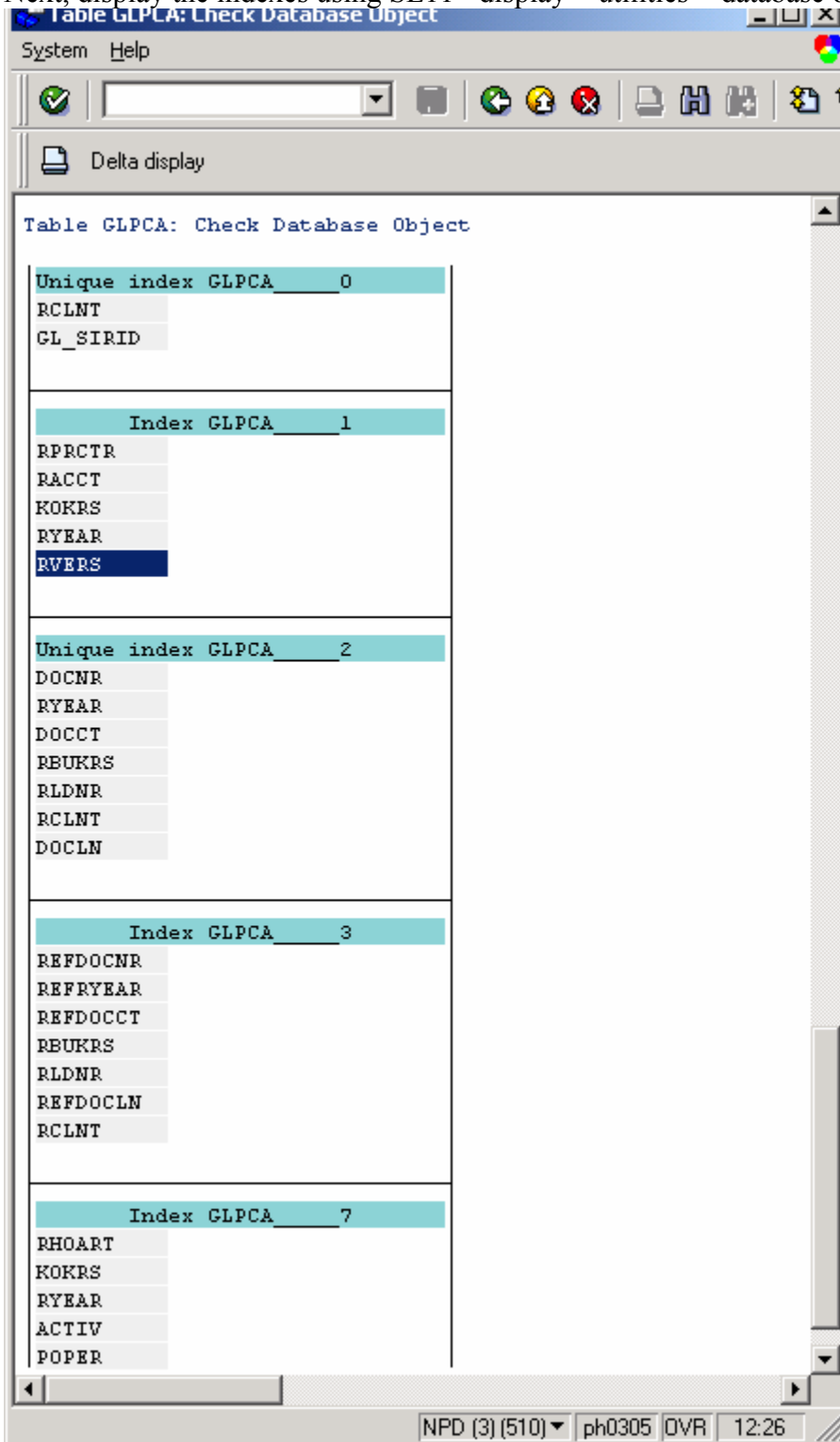
```

**Figure 124: GLPCA explain**

The access looks reasonable – 5 matching columns of 5 available on an SAP standard index. But we know a better index is possible, since Figure 121 showed that there were over 10 rows examined in the table, for every row processed, which shows that many rows were eliminated when examined in the table, rather than the index.



Next, display the indexes using SE11 > display > utilities > database object > check.

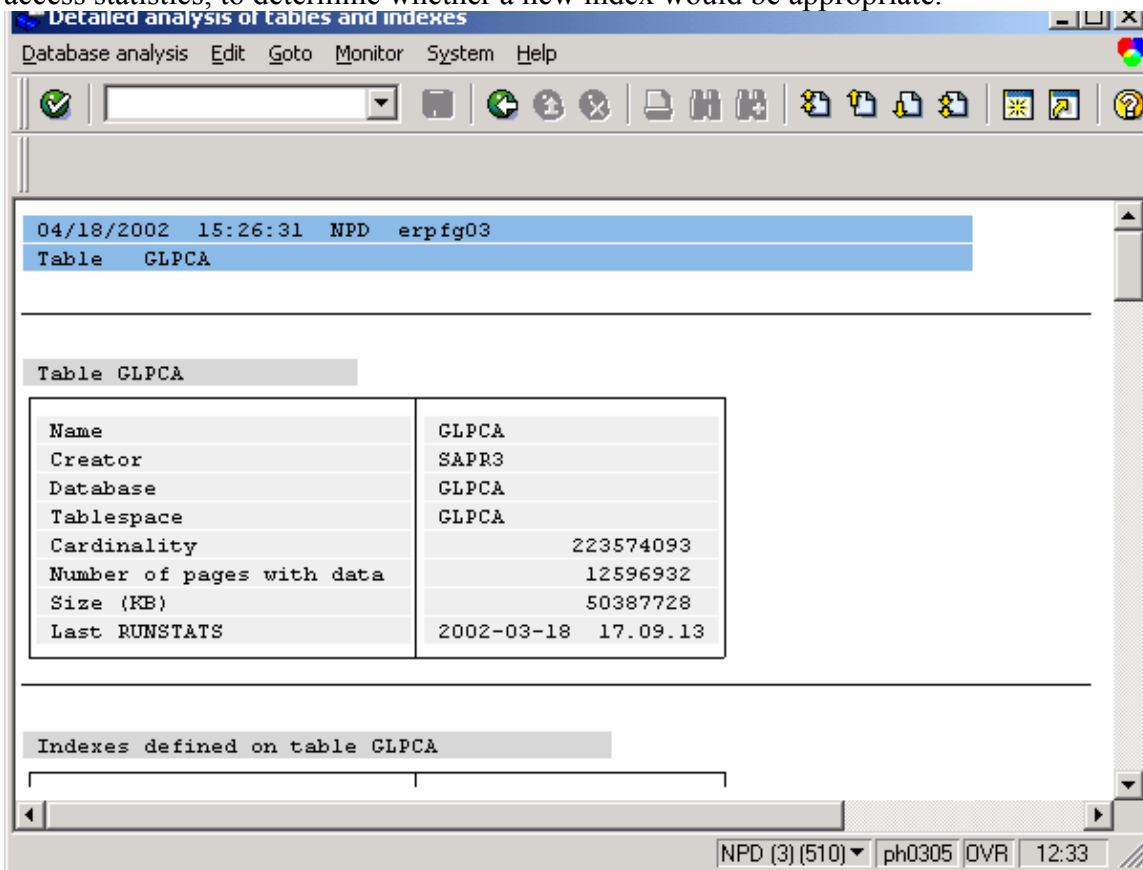


At this point, one would need to examine the cardinality of the predicate columns, and compare them to the available indexes, as was done on previous examples, to look for a better index. Examples of how to do this using the ST04 DB2 catalog browser are shown above. For this exercise, assume that the index being used is the best choice, according to the index cardinality data and matching columns.

Now, there are several choices:

- Add a new index
- Extend the current GLPCA~1 index with additional columns
- Do nothing

Check the size of the table using DB02 table detailed analysis, and how it is used using the ST10 table access statistics, to determine whether a new index would be appropriate.



**Figure 125: DB02 detailed table analysis of GLPCA**

The table has over 200,000,000 rows.

Look at ST10, to see how often it is changed.

TABLE	ABAP/IV Processor requests					DB activity		
	Changes/ Total (%)	Total	Direct reads	Seq. reads	Changes	Open	Fetch	Rows affected
<b>*Total*</b>		5899085200	4470503325	1287872481	140709394	173434248	237820691	1210211870
GLPCA	34.61	277,593	0	181,526	96,067	0	395,663	223281576
VBBE	0.38	136168878	0	135645284	523,594	0	2,957,011	131522770
EBAN	1.64	54977,551	441,932	53636,464	899,155	441,933	2,520,928	49493,721
RMMT	0.06	47948,999	183,071	47735,433	30,495	183,071	647,225	46773,979
S066	0.70	41253,475	0	40966,213	287,262	0	153,856	41155,139
RESB	0.27	29853,299	12	29772,607	80,680	12	57,279	29867,937
MDVV	5.69	29964,065	1	28258,790	1,705,274	1	64,927	29784,931
KMUV	0.03	25024,593	19224,059	5,793,830	6,704	19224,114	20443,688	23780,477

Figure 126: ST10 table call statistics for GLPCA

Note that changes are 34% of calls.

Last, check the size of the current index GLPCA~1 from DB02 detailed analysis.

04/18/2002 15:26:31 NPD erpfg03  
Table GLPCA

Size (KB)	2097360
Number of extents	1

Name	GLPCA__1
Creator	SAPR3
Database	GLPCA
Indexspace	GLPC13W#
Number of leaf pages	279831
Number of levels	4
Clusterratio	83
Date/time of space info	2002-04-18 00.00.00
Size (KB)	2097360
Number of extents	1

The index is over 2 GB.

Reviewing the three choices above:

- Add a new index that will filter the data better than the GLPCA~1 index.
  - Pro - it would speed up the program
  - Cons - 1) the table is relatively frequently changed, so a new index would slightly increase the cost of inserts and deletes, and possibly updates, and 2) the new index would likely be as large as index 1 (2GB) if not larger (added disk cost against the reduced CPU cost).
- Extend the currently used index GLPCA~1 so more rows can be eliminated via index access.
  - Pros - 1) it would speed up the program, and 2) it would not impact update processing (since we are not adding more indexes, just extending one).
  - Cons - 1) we try to avoid fiddling with SAP standard objects, and 2) the index would take more space (though not as much as adding a new index).
- Do nothing.
  - Pro - don't have to change anything
  - Con - don't get the performance improvement.

Depending on how critical the program is, we would probably take the “do nothing” option or “extend index GLPCA~1”. If the program runtime is not critical, then do nothing, and live with it. If program runtime is critical, then extend the current GLPCA~1 index. If the index is extended, one must do two things, 1) keep the order of the first 5 columns the same, so as not to cause changes to programs that currently use index GLPCA~1, and 2) always run KEYCARD RUNSTATS on the changed index, to gather the concatenated column cardinality statistics, so that the optimizer can make a good choice for programs that specify only the five (original) columns in the new index.

We evaluated the choice by looking at the expected improvement, compared to the impact on running programs, and the additional space requirements. Adding an index with a couple columns with high cardinality can be a good choice, as it offers efficient access, and does not take a lot of space. Adding indexes with many columns on large tables needs to be justified by a large improvement in an important business process, to offset the increased disk space utilization.

#### **8.4.9. Logical row lock contention**

This is common on statistics 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 are posted to a single “cost of goods sold” account (as we have seen done with SAP), then that row will become a constraint in the database when the transaction volume reaches a certain level. Below the critical level, it is not a problem.

The actions to fix row lock contention may require changes to SAP settings (“late exclusive material block”, “posting block”, etc) , ABAP coding (grouping changes for execution just before commit), and business definition in SAP tables (chart of accounts, statistics definitions, etc). 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 minimize 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.

Example 7.5.2 showed the tools that can be used in diagnosing locking problems.

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. In this case, a single row often has contention, but there is no database level tuning to fix it. As above, the fixes are application or workload configuration specific.

```

MEMBER: N/P
SUBSYSTEM: DBWO
DB2 VERSION: V5
ORDER: DATABASE-PAGESET
SCOPE: MEMBER
TO: NOT SPECIFIED
INTERVAL FROM: 09/16/99 10:48:28.35
TO: 09/16/99 10:55:22.29

```

DATABASE PAGESET	--- L O C K R E S O U R C E ---		TOTAL SUSPENDS	--SUSPEND REASONS--				R E S U M E		R E A S O N S			
	TYPE	NAME		LOCAL LATCH	GLOB. IRLMQ	S.NFY OTHER	NMNR	NORMAL AET	NMNR	TIMEOUT/CANCEL AET	NMNR	DEADLOCK AET	
ROW	PAGE=X'00100106'		343	343	0	0	0	343	0.268938	0	N/C	0	N/C
ROW	ROW =X'25'			0	0	0	0	0					
ROW	PAGE=X'00100106'		124	123	0	0	0	124	0.154899	0	N/C	0	N/C
ROW	ROW =X'26'			1	0	0	0	0					
ROW	PAGE=X'00100106'		174	174	0	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 127: 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 (which is usually a safe assumption) work on finding the level of parallelism where we had the maximal total throughput.

### 8.4.10. DB2 page latch contention

When multiple threads are simultaneously changing rows in a page, then page latch contention will occur. Page latch contention is rarely seen, except in large change intensive systems.

The two system indicators are ST04 “page latch” time being high, and change SQL statements in the ST04 statement cache with long elapsed time.

As in example 7.5.2, DB2PM suspension traces (IFCID 226,227) can be used to check the object with the page latch contention.

Here is a sample of a suspension trace showing page latch contention (name and type are PAGE, and suspension reason is OTHER).



SUBSYSTEM: DEW0 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

DATABASE PAGESET	--- L O C K R E S O U R C E ---		TOTAL SUSPENDS	--SUSPEND REASONS--				R E S U M E R E A S O N S			--- DEADLOCK --- AET	
	TYPE	NAME		LOCAL LATCH	GLOB. IRLMQ	S.NFY OTHER	NORMAL MMBR	TIMEOUT/CANCEL AET	MMBR AET	MMBR AET		
PAGE		PAGE=X'0030995F'	30	0	0	0	30	0.002531	0	N/C	0	N/C
		BPID=BP5		0	0	30						
PAGE		PAGE=X'00309960'	1	0	0	0	1	0.000056	0	N/C	0	N/C
		BPID=BP5		0	0	1						
PAGE		PAGE=X'0030997A'	2	0	0	0	2	0.001488	0	N/C	0	N/C
		BPID=BP5		0	0	2						
PAGE		PAGE=X'00309970'	1	0	0	0	1	0.000124	0	N/C	0	N/C
		BPID=BP5		0	0	1						
PAGE		PAGE=X'00309971'	2	0	0	0	2	0.002124	0	N/C	0	N/C
		BPID=BP5		0	0	2						
PAGE		PAGE=X'00309973'	1	0	0	0	1	0.000110	0	N/C	0	N/C
		BPID=BP5		0	0	1						
**	SUM OF COEPHO	**	4023	0	0	0	4023	0.025133	0	N/C	0	N/C
				0	0	4023						

Figure 128: DB2PM suspension page latch

Over 7 minutes elapsed time, there was only 101 seconds of delay (4023 suspensions \* 0.025 seconds) so this is not a major cause of delay, if there are many concurrent threads processing.

## 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 129: 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.)  $363,431 \text{ page in per hour} / 3600 \text{ seconds per hour} / 24 \text{ processors} / 0.40 \text{ utilization} = 10.5$  “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 history or ST06 current statistics. In the ST06 main panel, there is a current utilization display. ST06 > “detail analysis “ can be used to display hourly averages for the previous days. Figure 130 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 transferred/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 130: ST06 CPU constraint

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 /h	System calls /h	Context switches /h
	User	System	Idle			
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 131: 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.

### 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 132 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
<b>Nametab (NTAB)</b>					
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
<b>Program</b>					
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
<b>Tables</b>					
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 [%]	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 132: 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_SBI_00  0 DIA    58114 | running          Yes          1 010 XXX_BATCH          Roll Out
| des0101_SBI_00  1 DIA    58482 | running          Yes          1 010 XXX_BATCH          Roll Out
| des0101_SBI_00  2 DIA    52776 | running          Yes          1 010 XXX_BATCH          Roll In
| des0101_SBI_00  3 DIA    51198 | running          Yes          1 010 XXX_BATCH          Roll Out
| des0101_SBI_00  4 DIA    50934 | running          Yes          1 010 XXX_BATCH          Roll Out
| des0101_SBI_00  5 DIA    50122 | running          Yes          1 010 XXX_BATCH          Roll Out
| des0101_SBI_00  6 DIA    49708 | running          Yes          1 010 XXX_BATCH          Roll Out
| des0101_SBI_00  7 DIA    49570 | running          Yes          1 010 XXX_BATCH          Roll In
| des0101_SBI_00  8 DIA    43922 | running          Yes          1 010 XXX_BATCH          Roll Out
| des0101_SBI_00  9 DIA    17738 | running          Yes          1 010 XXX_BATCH          Roll Out
| des0101_SBI_00 10 DIA    17458 | running          Yes          1 010 XXX_BATCH          Roll In
| des0101_SBI_00 11 DIA    15432 | running          Yes          1 010 XXX_BATCH          Roll Out
| des0101_SBI_00 12 DIA    14242 | running          Yes          1 010 XXX_BATCH          Roll In
| des0101_SBI_00 14 DIA    54406 | running          Yes          1 010 XXX_BATCH          Roll Out
| des0101_SBI_00 15 DIA    28654 | running          Yes          1 010 XXX_BATCH          Roll Out
| des0101_SBI_00 16 ENQ    12594 | running          Yes          1 010 XXX_BATCH          Roll In
| des0101_SBI_00 19 DIA    43188 | running          Yes          1 010 XXX_BATCH          Roll In
| des0797_SBI_00  23 BTC    28776 | running          Yes          383 010 XXX_BATCH  SAPLPGRW  SAPLPGRW
| des0797_SBI_00  24 BTC    26936 | stopped ENQ      Yes          143 010 XXX_BATCH  SAPLSENA  SAPLSENA
| des0797_SBI_00  25 BTC    28578 | stopped CPIC    Yes          469 010 XXX_BATCH  SAPLSENA  CMSEND(SAP) / 85179598
| des0797_SBI_00  26 BTC    26556 | stopped CPIC    Yes          586 010 XXX_BATCH  SAPLSENA  CMSEND(SAP) / 83211579
| des0797_SBI_00  27 BTC    25804 | stopped ENQ      Yes          391 010 XXX_BATCH  SAPLSENA  SAPLSENA
| des0797_SBI_00  28 BTC    25592 | stopped CPIC    Yes          115 010 XXX_BATCH  SAPLSENA  CMSEND(SAP) / 82180132
| des0797_SBI_00  29 BTC    25270 | stopped CPIC    Yes          623 010 XXX_BATCH  SAPLSENA  CMSEND(SAP) / 82521911
| des08b3_SBI_00  11 BTC    36560 | running          Yes          73 010 XXX_BATCH          Sequential read
| des08b3_SBI_00  23 BTC    23718 | stopped ENQ      Yes          620 010 XXX_BATCH          Roll Out
-----

```

Figure 133: 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 134: 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 about 100 times more calls than the nearest buffer type.

```

-----
System          : ibmftaa8_FTA_00      Tune summary
Date & time of snapshot: 05/01/2001 14:06:58  Startup: 05/01/2001 08:31:22
-----

```

Buffer	Hitratio [%]	Allocated [kB]	Free space [kB]	Free [%]	Dir. size Entries	Free directory Entries	Free [%]	Swaps	Database accesses
<b>Nametab (NTAB)</b>									
Table definition	99.86	3,364	2,488	91.00	20,000	18,200	91.00	0	2,781
Field description	99.40	31,567	22,766	75.89	40,001	38,010	95.02	0	2,954
Short NTAB	99.91	4,067	2,362	94.48	40,001	39,181	97.95	0	820
Initial records	99.81	5,567	3,528	88.20	40,001	38,574	96.43	0	1,427
<b>Program</b>									
Program	99.12	329,809	103,116	31.73	40,625	39,142	96.35	0	4,449
CUA	99.49	5,000	3,802	80.81	2,500	2,288	91.52	0	212
Screen	99.51	19,532	18,080	94.09	4,500	4,289	95.31	0	215
Calendar	99.29	488	329	68.83	200	59	29.50	0	141
<b>Tables</b>									
Generic key	97.47	19,531	10,873	60.31	10,000	9,434	94.34	201	402,188
Single record	99.34	30,000	27,728	92.61	500	442	88.40	0	4
Export/import	85.52	4,096	3,133	98.46	5,000	4,928	98.56	0	0

Figure 135: ST02 generic buffer swapping

One can look at this problem in more detail by drilling into the “generic key” line in ST02, and 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.

```

-----
System          : ibmftaa8_FTA_00      Generic key buffered tables
Date & time of snapshot: 05/01/2001 14:08:12  System Startup: 05/01/2001 08:31:22
-----

```

Table	Buffer State	Buf key opt	Invali- dations	Total	ABAP/IV Processor requests	Seq. reads	Changes	DB activity	Rows affected
*Total*			46	12355,389	8,861,850	3,493,526	13	246,360	1,264,782
TVTA	error	gen	0	130,374	127,114	3,260	0	130,386	89,384
A040	error	ful	0	60,290	0	60,290	0	27,205	53,136
ZFV_DEPENDENCY	valid	ful	0	42,378	42,370	8	0	8,676	4,330
A957	error	gen	0	152,867	0	152,867	0	7,873	19,094

Figure 136: 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 SAPnote 3501 for information on how to interpret the state of tables in ST02.

### 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.

The screenshot shows the SAP Performance analysis: Table call statistics window. The window title is "Performance analysis: Table call statistics". The menu bar includes "Tune", "Edit", "Goto", "Monitor", "System", and "Help". The toolbar contains various icons for navigation and analysis. The main area displays the following information:

System: All servers  
 Not buffered tables  
 Time frame of analysis : 03/25/2002 - 04/01/2002

Table	ABAP/IV Processor requests				DB activity		
	Changes/ Total (%)	Total	Direct reads	Seq. reads	Changes	Calls	Rows affected
<b>*Total*</b>		971987980	895048496	61364,711	15574,773	157790132	551922573
ZBWEU_PROMOTION	0.00	17462,519	17462,512	7	0	34924,495	17462,552
KOMH	0.01	9,111,858	9,109,966	1,399	493	18222,465	9,135,042
EKPO	0.05	6,943,392	5,525,160	1,414,451	3,781	12486,271	13949,726
EKBE	0.10	6,405,583	231	6,399,105	6,247	6,456,768	14951,684
EKEZ	0.00	6,371,766	3	6,371,552	211	6,375,546	135,395
MARA	0.01	1,744,232	1,240,309	503,820	103	3,081,131	6,442,457
MARM	0.07	1,653,061	869,543	782,325	1,193	2,539,243	6,135,361
KNA1	0.00	1,193,270	1,180,277	12,973	20	2,400,652	3,198,064
VBUK	1.73	1,197,274	1,120,940	55,587	20,747	2,329,814	1,526,655
VBFA	0.53	2,192,024	43	2,180,419	11,562	2,204,716	4,261,695
EDIDS	2.07	2,139,954	0	2,095,656	44,298	2,141,603	6,407,272

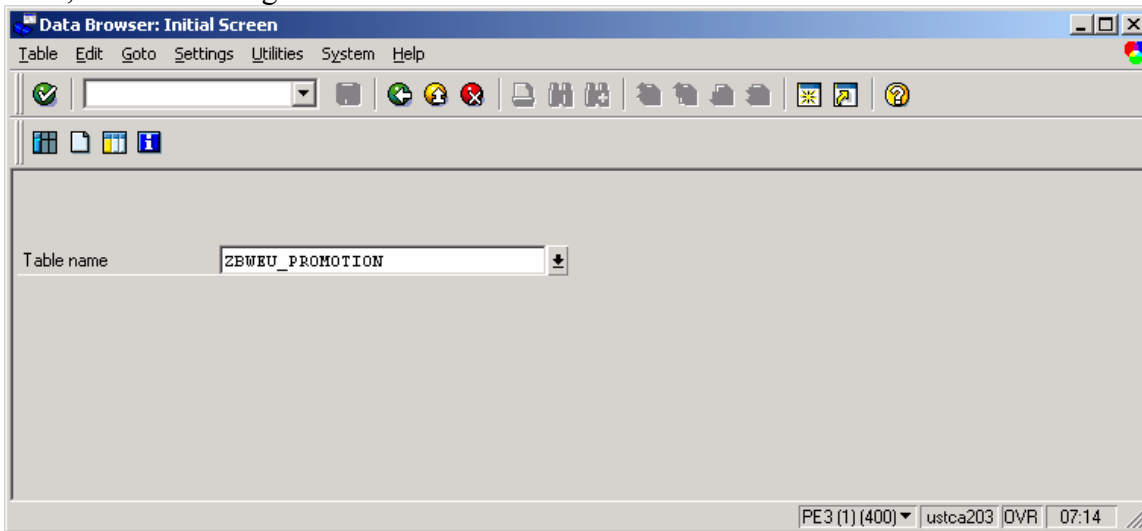
The status bar at the bottom right shows "PE3(1)[400] ustca203 OVR 07:16".

Figure 137: ST10 > not buffered, previous week, all servers > sort by calls

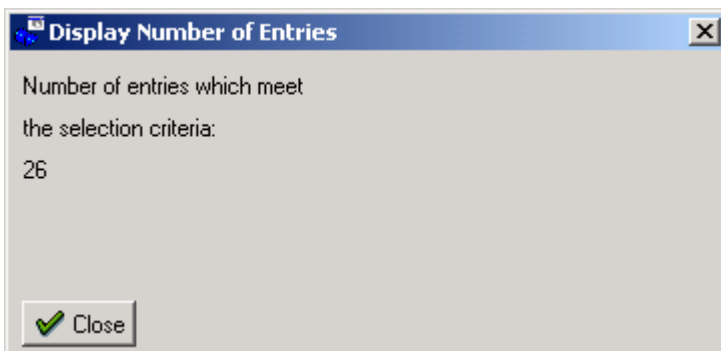
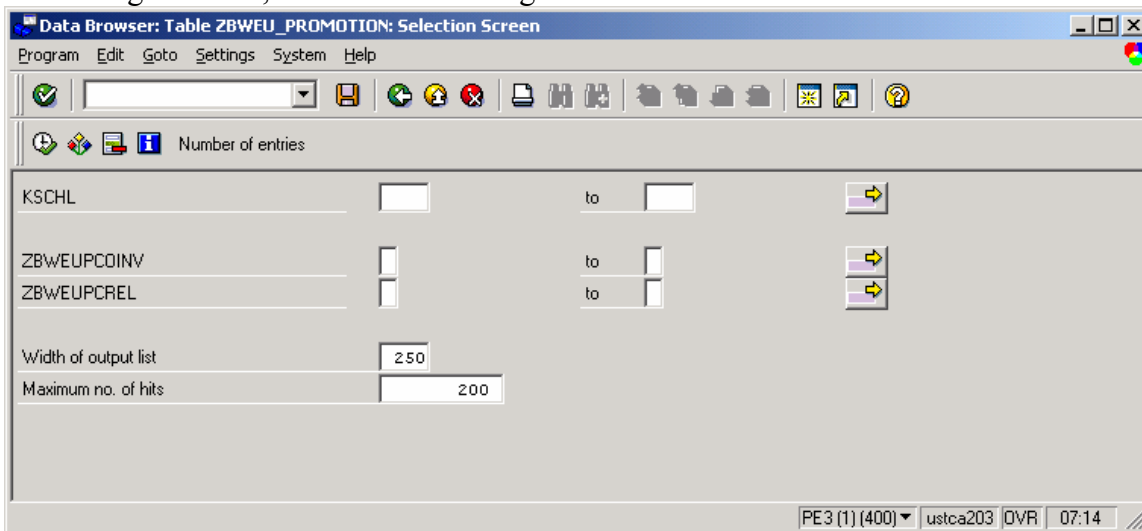
Here, note that the top entry has no changes. “Read only (for the most part)” is our first criteria for a candidate.



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



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.



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.

Performance analysis: Table call statistics

System: ustca203      Single record buffered tables  
Time frame of analysis : 04/01/2002 - 04/08/2002

Table	Buf key opt	Invalidations	ABAP/IV Processor requests				DB activity	
			Total	Direct reads	Seq. reads	Changes	Calls	Rows affected
<b>*Total*</b>		139	32411,212	32333,232	62,802	15,178	204,150	28896,790
TFDIR	sng	0	1,039,865	1,033,128	6,737	0	79,352	22279,095
IDOCREL	sng	0	52,530	0	39,980	12,550	53,195	51,547
AGR_DEFINE	sng	9	38,093	34,751	3,083	259	35,735	6,216,876
USREFUS	sng	0	93,914	88,054	5,598	262	6,626	5,743
DOKIL	sng	31	20958,252	20958,251	1	0	3,855	1,784
UTAB_USR10	sng	8	2,456	1,177	1,279	0	3,836	20,758
USRBF3	sng	13	89,988	89,305	0	683	3,722	1,731
TADIR	sng	7	9,954	9,832	2	120	2,697	60,788

PE3 (1) (400) | ustca203 | INS | 15:52

Figure 138: single record buffering on table accessed via select

In Figure 138 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 is ineffective. (Actually, IDOCREL has another problem too – it is frequently changed. It should not be buffered.)

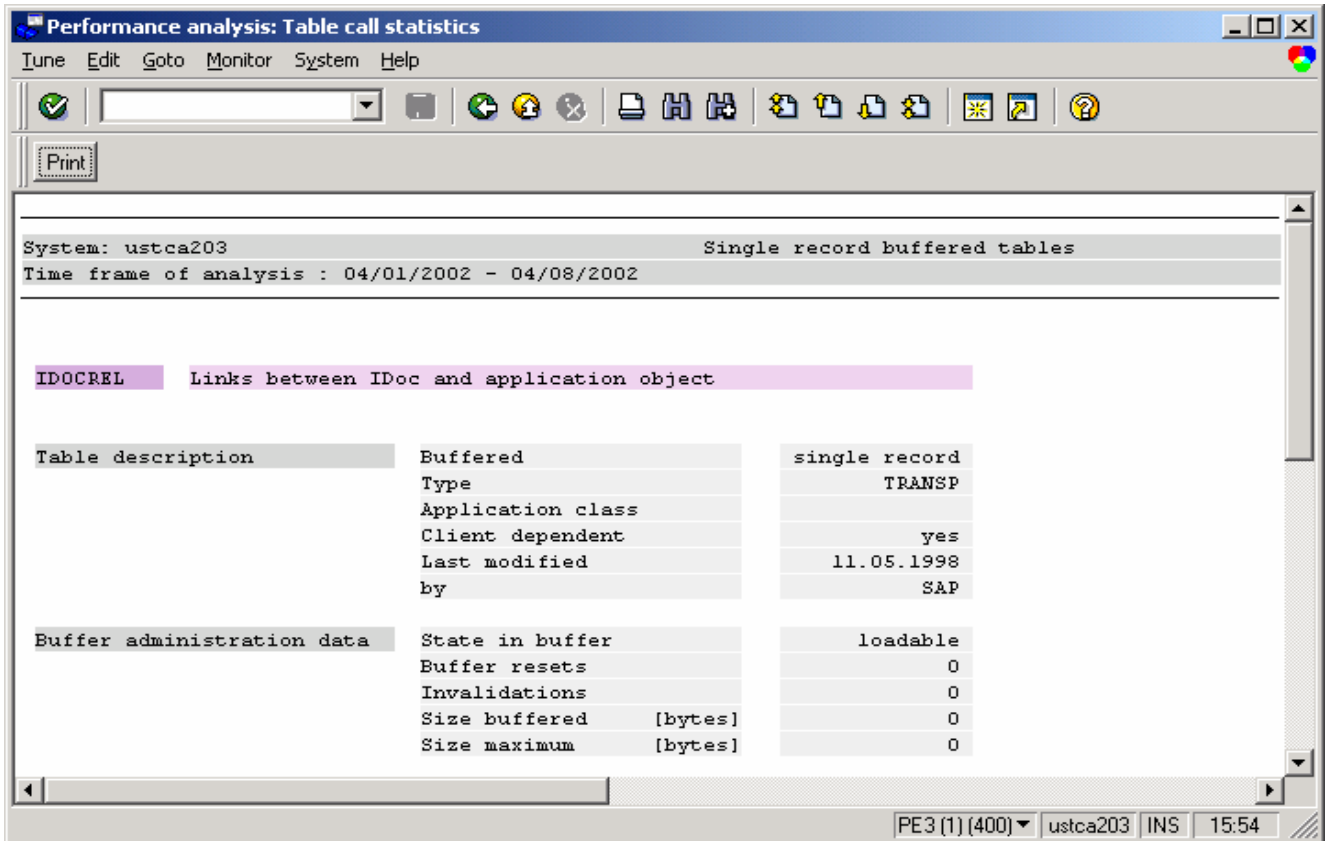


Figure 139: ST10 table details

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 SAPnotes, 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	ClieUser	Action	Table
10	DIA 70632	running	Yes						<no buff	000 SAPSYS	Direct read	NRIV
11	DIA 64766	waiting	Yes									
12	DIA 60848	waiting	Yes									
13	DIA 60564	waiting	Yes									
14	DIA 59182	waiting	Yes									
15	DIA 55944	running	Yes						RSMON000	440 IOC4779		
16	DIA 55638	waiting	Yes									
17	DIA 54966	stopped	CPIC	Yes				1	SAPMSSY1	440 T060504		
18	DIA 54300	waiting	Yes									
19	DIA 53246	waiting	Yes									
110	DIA 52622	waiting	Yes									
111	DIA 52392	stopped	CPIC	Yes				2	SAPMSSY1	440 T060504		
112	UPD 49918	running	Yes						SAPLBOIF	440 T060504	Sequential read	VEBATA
113	UPD 47376	running	Yes						RSML3000	440 T060504	Sequential read	VEBMOD
114	UPD 46500	running	Yes					1	RSML3000	440 T060504	Commit	
115	UPD 46186	running	Yes						RSML3000	440 T060504	Commit	
116	BTC 45558	running	Yes					5117	SAPLARFC	440 T060504	Insert	ARFCSSTATE
117	BTC 40806	stopped	NUM	Yes				967	SAPLSNR3	440 T060504		
118	BTC 39320	running	Yes			8		1	SAPLSNR3	440 T060504		
119	BTC 38332	running	Yes			8		5028	SAPLSNR3	440 T060504		
120	BTC 38066	running	Yes					2162	SAPLSZA0	440 T060504	Insert	ADRT
121	BTC 37794	running	Yes					5729	SAPLLE00	440 T060504	Insert	IFL0T
122	BTC 37408	running	Yes					6	SAPLEUD1	440 T060504		
123	BTC 36430	running	Yes					791	SAPLES05	440 T060504	Direct read	KNAL
124	SPO 80132	waiting	Yes									
125	UP2 79242	running	Yes					1	SAPLSCD0	440 T060504	Insert	CDCLS
126	UP2 73914	running	Yes						RSML3000	440 T060504	Delete	VEHDR

Figure 140: 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 141: 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 142, where NRIV updates take 56 ms, on the average.

Executions	Avg. elapsed time	Elapsed time	Statement text
99915	0.006	10:01.587	SELECT * FROM "EKPO" WHERE "MANDT" = ? AND
135	4.018	9:02.519	SELECT * FROM "SUNWIHEAD" WHERE "CLIENT" = ?
157473	0.002	8:12.713	INSERT INTO "VBOX" ( "MANDT" , "KAPPL" , "KOTABNR
601622	0.000	7:29.249	SELECT * FROM "EKBE" WHERE "MANDT" = ? AND
45	8.512	6:23.080	SELECT T_00 . "VBELN" FROM "VBRP" T_00 , "VBRK" T
182	1.854	5:37.459	SELECT * FROM "TFDIR" WHERE "FMODE" = ? ORD
350261	0.000	5:33.385	SELECT "VOLUM" , "VOLEH" FROM "LIPS" WHERE "MA
3981	0.073	4:53.633	SELECT "BERID" , "LGORT" , "CHARG" , "VBVYP" , "M
99711	0.002	4:49.841	SELECT * FROM "EBAN" WHERE "MANDT" = ? AND
4369	0.065	4:47.888	SELECT * FROM "EBAN" WHERE "MANDT" = ? AND "BANFN" =
6440	0.042	4:37.102	INSERT INTO "IDOCREL" ( "CLIENT" , "BOLE_A" , "DO
4483	0.056	4:15.535	SELECT * FROM "NRIV" WHERE "CLIENT" = ? AND

Figure 142: 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 143: 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 SAPnote 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 144: 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

#### 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.

SQL trace in SAP can also be a pointer to this problem, when the trace shows statements that intermittently take a very long time (can be hundreds of ms, or seconds) to complete.

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 1-3ms. 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	35,864	65	418	10,064	18.3
Direct read	258	63	63		1,050	4.1
Sequential read	247	34,955	2	378	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 145: 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 146: ST11 > display - ICLI network statistics in developer trace

In Figure 146, 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.

### 9.4.2. Lost packets example

Figure 147 is a trace from a 4.0 SAP DB2/390 system that was having packet loss due to incorrect switch settings. On this system, the retry time for lost packets was 10 seconds. To create a trace like this, run ST05, summarize the trace (ST05 > list trace > goto >summary), and sort by time.

SAP systems running TCP/IP do not have a fixed 10 second timer for retry, but will show similar behavior to what is shown in this trace -- that is there are intermittent statements that take much longer (50-100 times, or more) than normal.

PID	Pro	Cli	Time	Tcode/pr	Table	SQL op	Recs.	Time
00000458	DIA	040	12:57:20.536	VA01	D010L	SEL	1	11336.197
00000458	DIA	040	12:56:56.629	VA01	D021T	SEL	281	11238.554
00000458	DIA	040	12:57:32.353	VA01	D010Y	SEL	1	10518.399
00000458	DIA	040	12:57:08.717	VA01	D010L	SEL	1	10154.791
00000458	DIA	040	12:56:56.153	VA01	D021T	SEL	281	474.638
00000458	DIA	040	12:56:50.192	ST10	MONI	SEL	30	343.094
00000458	DIA	040	12:57:43.916	VA01	D301T	SEL	324	320.546
00000458	DIA	040	12:57:43.392	VA01	ATAB-((((((((	SEL	50	222.152
00000458	DIA	040	12:57:43.620	VA01	D342L	SEL	1	177.098
00000401	DIA	040	12:57:46.912		D010INC	SEL	6	172.120
00000458	DIA	040	12:56:49.960	ST10	D021T	SEL	54	155.032
00000458	DIA	040	12:56:37.235	AL08	D010T	SEL	1	152.952
00000458	DIA	040	12:56:49.760		D010TAB	SEL	11	151.362
00000458	DIA	040	12:56:36.823		D010INC	SEL	5	144.052
00000401	DIA	040	12:57:47.085		D010TAB	SEL	7	135.429
00000401	DIA	040	12:57:46.208	ARTO	D010SINF	SEL	1	133.332
00000401	DIA	040	12:57:46.385		D010S	SEL	1	133.027
00000458	DIA	040	12:57:43.094	VA01	D010T	SEL	1	124.763
00000458	DIA	040	12:56:36.976		D010TAB	SEL	8	112.983
00000401	DIA	040	12:57:46.640		D010S	SEL	1	111.745

Figure 147: ST05 sorted summary with lost packets

## 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 I and RMF 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  $100 * (\text{random getpages} - \text{synchronous read random}) / (\text{random getpages})$ , 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-referenced (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 occurs frequently and impacts performance, use DB2 traces with IFCID 125 to find the cause. Then evaluate ways to influence the access path, such as reoptimization at execution time.
- **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.

Recent versions of SAP and DB2 report RID failures in the ST04 statement cache as part of the statement statistics, so one does not have to use IFCID 125 to find RID processing failures.

### 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 OS/390 page between CS (central store) and ES (expanded store), or disk (aka aux storage).

Use the customary OS/390 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 OS/390 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 OS/390 to take ES pages from hiperpools. You can also see the impact of OS/390 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 OS/390 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.

PAGING ACTIVITY										PAGE	2	
OS/390		SYSTEM ID TCPI		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
		WRITTEN TO		READ FROM		MIGRATED		FREED				
		EXP STOR		EXP STOR		EXP STOR		MIGRATION				
TOTAL	RT	225.78	78.68	43.91	54.72	*----- EXPANDED STORAGE FRAME COUNTS -----*		MIN	MAX	AVG		
PAGES	%	100.0%	100.0%	100.0%		255,380	257,898	257,579				
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 148: 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 it is more efficient for DB2 to do page movement between CS and ES by moving between bufferpool and hiperspace, 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 hiperspaces, 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
INSTALLED   ONLINE
-----
925696     925696
HIGH UIC   MIGR AGE
MIN        5     99373
MAX       70   104646
AVG      33.1 102007.9
-----
          WRITTEN TO   READ FROM   MIGRATED   FREED
          EXP STOR    EXP STOR    EXP STOR   WITHOUT
TOTAL    RT          421.52     399.90     0.00       0.00
PAGES   %          100.0%    100.0%    100.0%
HIPERSPACE RT      0.00      0.01      0.00
PAGES   %          0.0%     0.0%     0.0%
VIO     RT          0.00      0.00      0.00
PAGES   %          0.0%     0.0%     0.0%
SHARED  RT          0.00      0.06
PAGES   %          0.0%     0.0%
-----
          *----- EXPANDED STORAGE FRAME COUNTS -----*
          MIN          MAX          AVG
TOTAL    359,382      374,664      369,330
HIPERSPACE 132,000      132,000      132,000
VIO         8          10           9
SHARED
PAGES
    
```

Figure 149: RMF I Paging report – CS constraint

### 9.6.3. CPU constraint

See the example in 8.2.3.

### 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 OS/390 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
|                          75,593           Direct reads                  3,215,016
| Dialog steps                                     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 150: 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 is 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 151: ST04 times long service task switch

Now, go to OS/390 RMF III. Check DEV to see device delays.

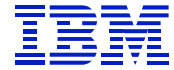
```

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

Jobname  Service  DLY USC CON  -----  Main Delay Volume(s) -----
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 152: 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 153: 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 OS/390 cause.

## Estimating the impact of fixing problems

### 9.7. *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.

#### 9.7.1. Estimate system impact of inefficient SQL

By comparing the ST04 statement cache statistics and the ST04 bufferpool statistics over an interval, one can estimate the impact of inefficient SQL on the entire system.

In this example, we have stopped IFCID 318 and restarted it, which resets the statement cache statistics, so that the statement counters are gathered over a known interval. Use ST04 “subsystem activity” to reset subsystem statistics at the start of the interval, and then use “since reset” to report on bufferpool and SQL activity since the start of the interval.

DB2 CPU usage is related to the number of getpage operations performed. In general, a statement that performs many DB2 getpages to return a result will use more CPU than a statement that performs fewer getpages – searching the additional pages required additional CPU. Use the getpage sort in ST04 statement cache, and compare the total getpages performed when executing a statement with the total number of getpages performed by DB2 in the same interval. Look for statements which are inefficient, and which perform a significant percentage of total DB2 getpages

Use ST04 > “subsystem activity” > “reset” to reset the statistics at 13:48. At the same time, stop and restart IFCID 318 from ST04 “DB2 commands”.

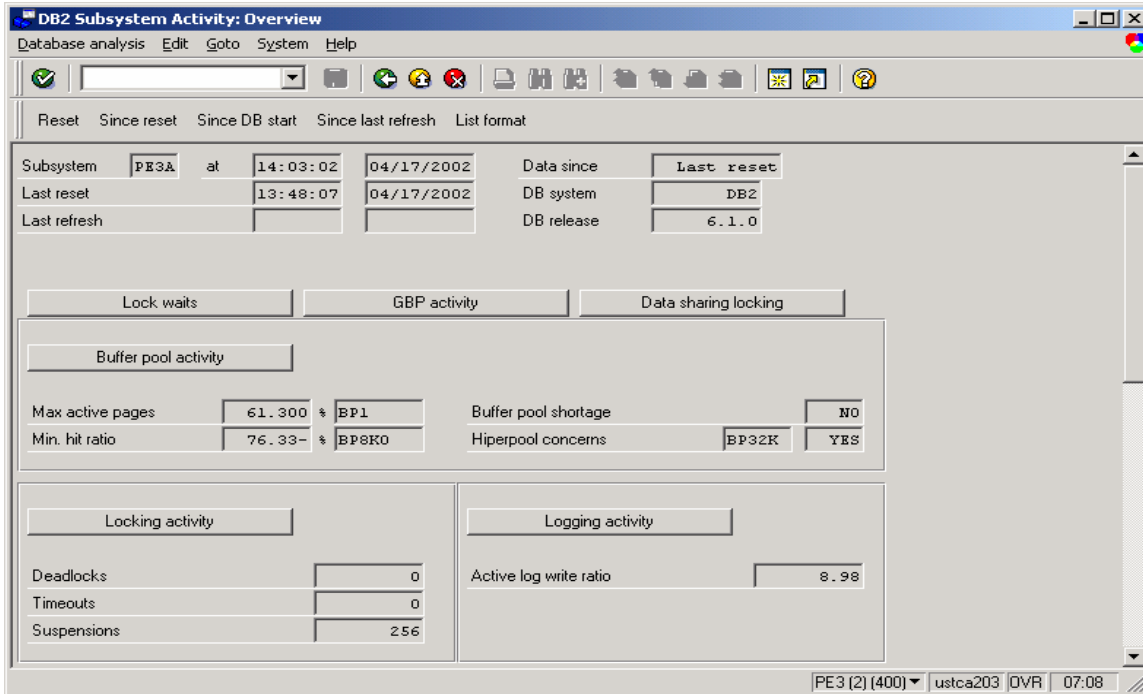


Figure 154: ST04 > DB2 subsystem activity

In DB2 subsystem activity, after a while, use “since reset” > “list format” to list the subsystem statistics over the interval.

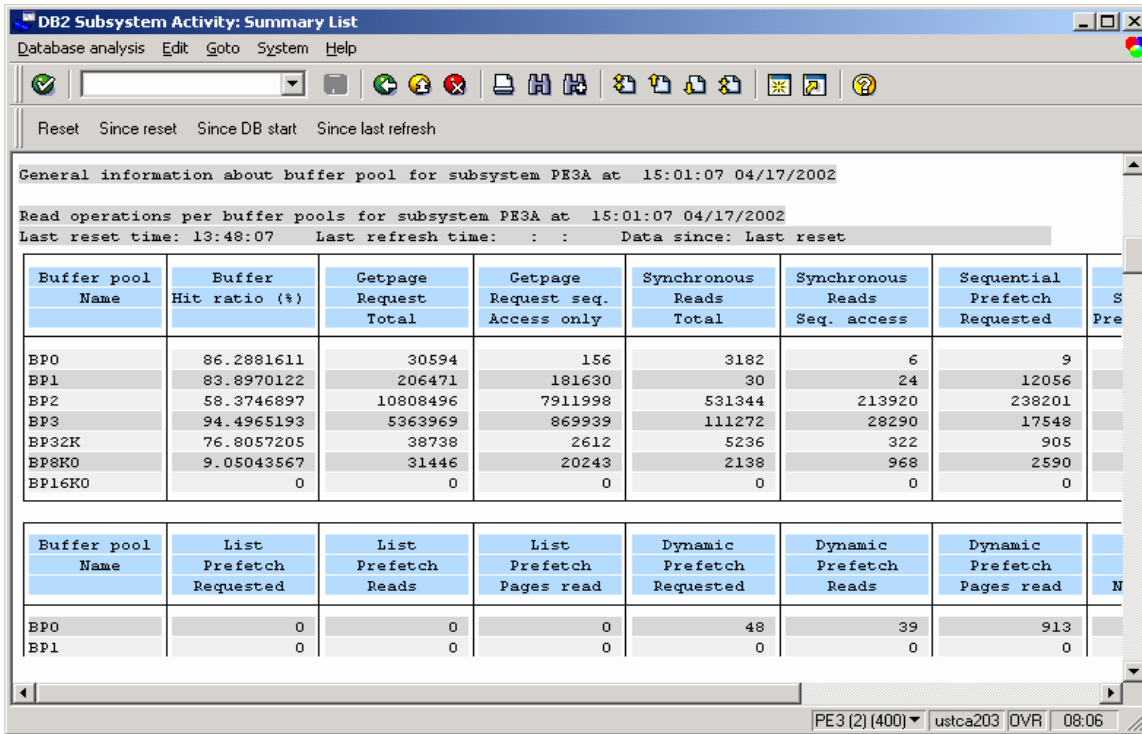
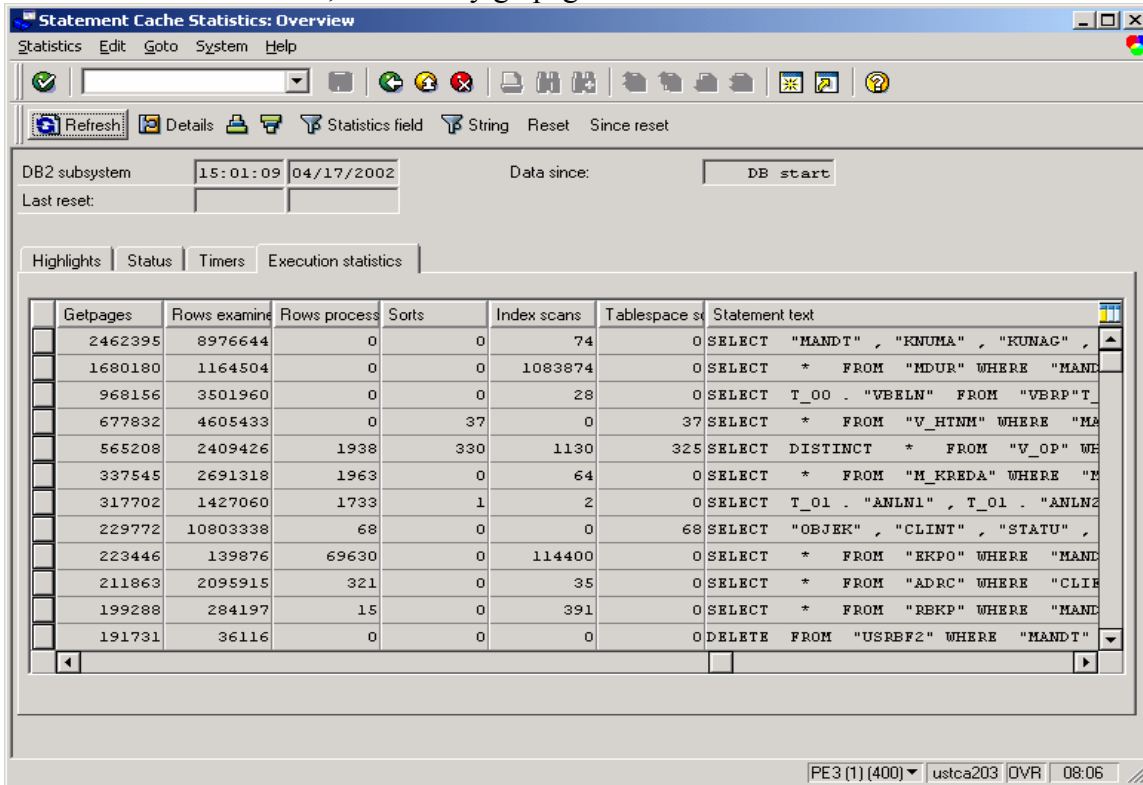


Figure 155: ST04 subsystem activity over interval

Display ST04 statement cache statistics. Since they were reset at the start of the interval via stop and restart of IFCID 318, this example does not use reset and “since reset” for statement cache. Use the “Execution statistics” tab, and sort by getpages.



**Figure 156: ST04 cached statement statistics over interval sorted by getpages**

Now, compare the total getpages over the interval from Figure 155 (16,300,000) with getpages performed by inefficient statements at the top of the statement cache list. Note that the four top statements return few rows compared to rows examined or getpages, so they may be inefficient and candidates for improvement.

The next step is to check the per-execution statement cache counters on the “Execution statistics” tab, which are not shown here. If these statements perform hundreds or thousands of getpages per execution, and return few rows on each execution, then they are candidates for improvement. If they perform few getpages per execution, and return few rows, they are already efficient. Use the analysis process outlined earlier (check available indexes, check catalog statistics, etc), to determine the cause of the inefficient access, and then take action based on that cause.

The two statements at the top of the list are performing 15% and 10%, respectively, of all the getpages done by DB2 during the interval. If we could improve the efficiency of these statements, then there would be a reduction in CPU usage, and there would very likely also be a reduction in I/O activity, as reducing getpages would reduce memory pressure on the bufferpools.

A rule-of-thumb is that inefficient statements that consume more than 5% of total getpages are high impact, and should be addressed promptly. Fixing several statements that consume more than 1% of

total getpages can, taken together, also have a measurable impact on the system performance, and they should also be addressed.

### 9.7.2. 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.

## 9.8. *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.



### 9.9. STAT- evaluating performance in an identified program

Using a high performance network such as Gigabit Ethernet, if the data can be accessed via efficient indexes, sequential reads take at most a few ms per row, direct reads are generally 1-3 ms per row, and change SQL takes a few ms per row. In cases where the direct read and sequential reads are often read from buffers on the application server, or where the sequential reads are array operations reading many rows per request, one can expect less than one ms per row.

When evaluating a STAT record, and to estimate what the performance would be with efficient access to the data (which is not always possible), one can use the ROTs above to create an estimated improvement.

This is the STAT record from the example in section 8.4.5, an I/O constraint on the disks with the PROP table and indexes.

End time	Tcod	Program	T	Scr.	Wp	User	Response time(ms)	Memory used(kB)	Wait time(ms)	CPU time(ms)	DB req. time(ms)	Load/Gen time(ms)	kBytes transfer	Phys. db changes
17:48:47		ZZRMPCOG B			8	APROGRM	12527361	1,732	0	1999,310	1463190	33	35872.9	1,867
18:03:28		ZZRMPCOG B			24	APROGRM	1939412	0	0	646,620	1272918	31	28905.4	102

Analysis of time in work process					
CPU time	646,620 ms	Number	Roll ins	0	
RFC+CPIC time	0 ms		Roll outs	1	
			Enqueues	2	
---Response time-----1939412 ms---					
Wait for work process	0 ms	Load time	Program	26 ms	
Processing time	666,458 ms		Screen	1 ms	
Load time	31 ms		CUA interf.	4 ms	
Generating time	0 ms				
Roll (in+wait) time	0 ms	Roll time	Out	19 ms	
Database request time	1272918 ms		In	0 ms	
Enqueue time	5 ms		Wait	0 ms	

Analysis of ABAP/4 database requests (only explicitly by application)					
Database requests total	307,042	Request time	1,272,918 ms		
		Matchcode time.	0 ms		
		Commit time	17 ms		
Requests on T??? tables	0	Request time	0 ms		

Type of	Database Requests	Database rows	Requests to buffer	Database calls	Request time(ms)	Avg.time per req.
ABAP/4 request	Requests	rows		calls	time(ms)	per req.
Total	307,042	118,331	291,197	16,674	1,272,918	4.1
Direct read	297,208	8,966	288,241		70,074	0.2
Sequential read	9,730	109,263	2,956	16,572	1,201,045	123.4
Update	4	2		2	103	25.8
Delete	0	0		0	0	0.0
Insert	100	100		100	1,679	16.8

Figure 157: STAT record with slow database request time

Now look at what the impact of slow I/O is. Check the relative amounts of database request time and CPU time in Figure 157. The ratio is about 2 to 1. Thus, the improvement opportunity in the database request time is about 66% of the elapsed time. Calculate the average time per row from the database

requests stanza -- 1201045 ms / 109263 rows = 11ms per row. 11 ms per row is slow. In general, sequential read “per row” access times for well-indexed data will be just a few ms per row, at the most. Efficiently indexed array operations such as seen here (109263 rows returned in 9730 requests for an average of 10 rows per request) are often less than one ms per row. Relieving the I/O constraint on this job should cut at least  $\frac{1}{2}$  of the database request time out of the job, and reduce program runtime by at least 30% ( $\frac{1}{2}$  of the 66% opportunity in DB request time). As shown in section 7.4.4, there are always exceptions to the ROTs, so improvement may vary.

## 10. How to tell when you are making progress

### 10.1. SAP

Normally, it is best to view improvement from the application, by using STAT or ST03 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 10.2

### 10.2. DB2 and S/390

Using DB2 or OS/390 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.

## 11. Appendix 1: summary of performance monitoring tools

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

### 11.1. SAP

#### 11.1.1. DB02

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)

#### 11.1.2. SE11

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
- 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.
  - The need to do “where used” will go away with DB2 V7 and SAP 6.20, when statements in ST04 statement cache will have a marker with the ABAP program name.

#### 11.1.3. SE30

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

#### 11.1.4. SM12

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 tools such as OMJI, “late exclusive material block”. There are different SAP settings for different parts of the business processes.
- High counts of error can point to a problem where the enqueue table is too small. Compare “peak util” with “granule arguments” and “granule entries” to check for the table filling.

**11.1.5. SM50**

SM50 is an overview of activity on an SAP instance. If many processes are doing the same thing (e.g. access same table, ENQ, CPIC, etc), this can point to where further investigation is required.

**11.1.6. SM51**

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.

**11.1.7. SM66**

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

**11.1.8. STAT**

Displays STAT records for a single SAP instance.

**11.1.9. STAD**

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.

**11.1.10. ST02**

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

**11.1.11. ST03**

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.

**11.1.12. ST04**

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.

**11.1.13. ST05**

ST05 is one of the most important tools for SAP performance, among its functions are:

- Trace calls to database server to check for inefficient SQL when program is known.
- Compress and save SQL traces for regression testing and comparisons.
- Trace RFC, enqueue, and locally buffered table calls.

**11.1.14. ST06**

Display OS level stats for the application server – paging, CPU usage, and disk activity.

**11.1.15. ST10**

ST10 is used to monitor table activity, and table buffering in SAP on the application server:

- Check for tables that are candidates for buffering in SAP
- Check for incorrectly buffered tables

**11.1.16. RSINCL00**

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

**11.1.17. SQLR0001**

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 which issued the problematic SQL.

**11.2. OS/390****11.2.1. 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.

**11.2.2. RMF II**

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

**11.2.3. 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

### **11.3. DB2**

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

#### **11.3.1. 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.
- Traces of specific IFCIDs are very useful for diagnosing specific 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.

## 12. Appendix 2: Reference Materials

### 12.1. *SAP Manuals*

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

### 12.2. *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”

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)



Figure 1: Sample STAT record.....	9
Figure 2: STAT database request with time per request.....	11
Figure 3: STAT database request time with time per row.....	12
Figure 4: stat/tabrec data.....	14
Figure 5: rsdb/stattime time statistics in ST10.....	14
Figure 6: STAT record with low CPU time.....	18
Figure 7: STAT record with high CPU time.....	19
Figure 8: STAT RFC detail.....	20
Figure 9: STAT wait for work process – symptom of SAP roll area overflow.....	20
Figure 10: STAT slow insert causes wait time.....	21
Figure 11: STAT record with CPU corresponding to Processing time.....	22
Figure 12: Processing time shows missing time in SAP.....	23
Figure 13: Processing time containing GUI time.....	23
Figure 14: processing time with GUI time removed.....	24
Figure 15: STAT high load time.....	25
Figure 16: STAT roll (in+wait) GUI time.....	25
Figure 17: STAT roll-in.....	26
Figure 18: ST06 > detail analysis > top CPU - showing processor constraint.....	28
Figure 19: SM50 showing ENQ wait.....	28
Figure 20: SM51 > Goto > queue information - display of queues on SAP central instance.....	29
Figure 21: STAT long total and average enqueue times.....	30
Figure 22: SM50 display on central instance showing Sem 26 (ENQ) wait.....	30
Figure 23: ST06 > detail analysis > top CPU - no processor constraint.....	31
Figure 24: ST06 > detail analysis > disk - high I/O activity on UNIX disk.....	31
Figure 25: filemon displays active filesystems.....	32
Figure 26: STAT with long GUI time.....	33
Figure 27: MIRO ST05.....	35
Figure 28: ST05 selection screen.....	36
Figure 29: MIRO ST05 trace list selection screen.....	37
Figure 30: ST05 > list trace - slow RBKP.....	38
Figure 31: ST05 RBKP Explain.....	38
Figure 32: Statement text with parameter markers – ST05 “replace vars”.....	40
Figure 33: ST04 cached statement statistics with RBKP statement.....	40
Figure 34: ST05 display of SQL statement with parameter values.....	41
Figure 35: ST05 source display of RBKP select with SAP dynamic SQL.....	42
Figure 36: ST04 DB2 catalog browser to query catalog statistics.....	43
Figure 37: MIRO example - check column cardinality via catalog browser.....	43
Figure 38: MIRO example - query to display indexes on RBKP.....	44
Figure 39: MIRO example - columns in indexes on RBKP.....	44
Figure 40: MIRO example - query to check index cardinality.....	45
Figure 41: RBKP indexes and cardinality.....	45
Figure 42: SE11 display index definition.....	47
Figure 43: STAT record for ME21.....	48
Figure 44: Summarized ST05 SQL trace with slow KSSK.....	49
Figure 45: ST05 trace list with slow KSSK.....	49

Figure 46: ST05 explained KSSK statement .....	50
Figure 47: KSSK statement from ST04 cached statement statistics .....	50
Figure 48: ST05 source display - ABAP selecting KSSK .....	51
Figure 49: ST05 display KSSK statement with variables .....	52
Figure 50: SE16 display table contents .....	53
Figure 51: KSSK check indexes and columns .....	54
Figure 52: KSSK indexes and columns .....	55
Figure 53: KSSK check index cardinality .....	55
Figure 54: KSSK index cardinality .....	56
Figure 55: KSSK check cardinality of columns in KSSK~N1 .....	56
Figure 56: MB51 - ST05 summarized trace .....	58
Figure 57: MKPF MSEG explain .....	60
Figure 58: MB51 predicate cardinality .....	61
Figure 59: MKPF MSEG query index statistics .....	63
Figure 60: MKPF MSEG index statistics .....	63
Figure 61: MKPF MSEG query table statistics .....	64
Figure 62: MKPF MSEG table statistics .....	64
Figure 63: REAPRIN0 STAT .....	66
Figure 64: DB2PM LOCKING REPORT .....	68
Figure 65: TST01 - Select starting point in summarized ST05 trace .....	70
Figure 66: SM50 display .....	74
Figure 67: DB2 time categories .....	78
Figure 68: ST04 global times .....	79
Figure 69: Good ST04 times .....	82
Figure 70: ST04 with long total “other read suspension” .....	83
Figure 71: ST04 times with high “Not attrib. in DB2” .....	84
Figure 72: OS07 - overview of DB server performance metrics .....	85
Figure 73: RMF III SYSINFO of 900 second interval .....	86
Figure 74: ST04 cached statement statistics sorted by getpages – execution statistics .....	88
Figure 75: ST04 cached statement statistics sorted by getpages – highlights .....	89
Figure 76: LIPS with index screening .....	91
Figure 77: SE11 to initiate “where used” .....	92
Figure 78: SE11 “where used” object selection .....	93
Figure 79: SE11 “Search area” popup .....	94
Figure 80: SE11 set development class in search area .....	94
Figure 81: SE11 hit list .....	95
Figure 82: SE11 where used expanded hit list .....	96
Figure 83: SE11 find to locate search string .....	96
Figure 84: SE11 found lines .....	97
Figure 85: SE11 found program .....	97
Figure 86: “details” display of M_VMCFB statement from ST04 cached statement .....	98
Figure 87: ST04 cache “details” display of execution statistics .....	99
Figure 88: Explained statement from ST04 cached statement details .....	100
Figure 89: VBRK column cardinality statistics .....	102
Figure 90: VBRP VBRK join .....	107
Figure 91: DB02 main screen .....	110

Figure 92: DB02 table detailed analysis - select table.....	110
Figure 93: DB02 detailed analysis for VBRK.....	111
Figure 94: DB02 detailed analysis for VBRP.....	111
Figure 95: ST04 statement cache for BDCPV.....	115
Figure 96: BDCPV statement.....	116
Figure 97: BDCPV explain.....	118
Figure 98: BDCPV query for cardinality of predicate columns.....	119
Figure 99: Column cardinality for BDCP and BDCPS.....	119
Figure 100: BDCPV query indexes on tables.....	120
Figure 101: BDCP and BDCPS indexes and columns.....	121
Figure 102: BDCPS table columns.....	122
Figure 103: BDCPS KEYCARD statistics query.....	123
Figure 104: BDCPS KEYCARD statistics.....	124
Figure 105: ST04 cached statement statistics with long select times on PROP.....	125
Figure 106: ST04 statement cache “details two” with PROP long elapsed time.....	126
Figure 107: RMF III DEV report for 300 second interval.....	126
Figure 108: RMF III DEVR report.....	127
Figure 109: RMF III DSNV report.....	127
Figure 110: RMF III DSNV report.....	128
Figure 111: ST04 cached statement highlights with index screening.....	129
Figure 112: ST04 cached statement execution statistics with index screening.....	129
Figure 113: FOR ALL ENTRIES.....	132
Figure 114: MDUP statement in ST04.....	133
Figure 115: MDUP per statement statistics in ST04.....	134
Figure 116: MDUP statement.....	134
Figure 117: MDUP explain.....	135
Figure 118: DB02 detailed analysis.....	136
Figure 119: DB02 detail analysis to show catalog statistics.....	138
Figure 120: MDUP statement statistics after runstats.....	140
Figure 121: ST04 statement cache screen 1 – GLPCA.....	142
Figure 122: ST04 statement cache screen 2 - GLPCA.....	142
Figure 123: GLPCA statement.....	143
Figure 124: GLPCA explain.....	144
Figure 125: DB02 detailed table analysis of GLPCA.....	146
Figure 126: ST10 table call statistics for GLPCA.....	147
Figure 127: DB2PM LOCKING REPORT for GLT0.....	149
Figure 128: DB2PM suspension page latch.....	150
Figure 129: ST06 > detail analysis > previous hours memory - high paging.....	154
Figure 130: ST06 CPU constraint.....	155
Figure 131: ST06 > detail analysis > previous hours CPU - CPU constraint.....	156
Figure 132: ST02 roll area over-committed.....	157
Figure 133: SM66 roll in and roll out.....	158
Figure 134: ST06 > detail analysis > disk - high I/O activity on ROLL area.....	158
Figure 135: ST02 generic buffer swapping.....	159
Figure 136: ST02 > drill into generic key > buffered objects.....	159
Figure 137: ST10 > not buffered, previous week, all servers > sort by calls.....	160

Figure 138: single record buffering on table accessed via select.....	162
Figure 139: ST10 table details .....	163
Figure 140: SM50 “stopped NUM” and Sem 8 .....	164
Figure 141: NRIV row-lock contention on 4.0 ST04 statement cache.....	164
Figure 142: NRIV row-lock contention on 4.6 ST04 statement cache.....	165
Figure 143: ST02 > detail analysis > number ranges - number range statistics .....	166
Figure 144: ST02 >detail analysis > semaphores - semaphore statistics.....	167
Figure 145: STAT record with slow change SQL .....	169
Figure 146: ST11 > display - ICLI network statistics in developer trace.....	169
Figure 147: ST05 sorted summary with lost packets.....	170
Figure 148: RMF I Paging report – ES constraint .....	175
Figure 149: RMF I Paging report – CS constraint.....	176
Figure 150: ST03 with long change and commit time.....	177
Figure 151: ST04 times long service task switch .....	177
Figure 152: RMF III DEV report.....	177
Figure 153: RMF III DSNV report .....	178
Figure 154: ST04 > DB2 subsystem activity.....	180
Figure 155: ST04 subsystem activity over interval .....	181
Figure 156: ST04 cached statement statistics over interval sorted by getpages .....	182
Figure 157: STAT record with slow database request time.....	185