MaxDB 7.6
Performance Analysis

Werner Thesing

Performance Analysis: Tools

x_cons
- shows current DB activity (snapshot)

Database Analyzer
- detects possible bottlenecks
- collects and stores data at given intervals

Diagnostic Monitor (Command Monitor)
- Lists single long running SQL commands

Diagnose Analyze (Resource Monitor)
- Displays accumulated data for all SQL commands

Performance Analysis Tools

MaxDB provides various tools and methods for the analysis of performance bottlenecks and monitoring current database activities. Some of these tools were originally developed only for testing and analysis in MaxDB development, but can also be used by experienced database administrators for performance analysis.

The following are of particular importance for performance analysis:

- The **x_cons** console for monitoring current operations
- The **Database Analyzer** program for analyzing performance bottlenecks
- The diagnostic function **DIAGNOSE MONITOR** for identifying long-running or poorly-processed SQL statements
- The diagnostic function **DIAGNOSE ANALYZE** for displaying information about all current SQL statements

x_cons and Database Analyzer are stabd-alone programs and are called from the operating system command line. DIAGNOSE MONITOR is a part of the core functions of MaxDB.

In SAP WebAS, all functions and results can be controlled and analyzed using transaction DB50 => Current Status or DB50 => Problem Analysis. Required parameter settings, if any, are menu-driven.

DB-Console: x_cons

Database console x_cons features:

- process overview
- configuration overview
- observing session activities and wait states
- watching I/O activities and wait queues
- measuring of detailed task specific times

Call:

- x_cons <serverdb> <command> [<interval>] [<repeat>]
  - e.g. x_cons E30 show active 10 6
  - advantage: deltainformation using ‚interval' and ‚repeat'
- dbmcli –d … -u ... [–n <node>] db_cons <command>
  - advantage: works per remote connection to database host

DB Console x_cons

The database console **x_cons** gives you a quick overview of the operating system resources that the database system is using, the distribution of the database session among the operating system threads, and the status of the active database sessions. You can also use other functions that are intended mainly for support employees and developers.

Start on shell level: x_cons <dbname > <command> [<interval>] [<repeat>]

x_cons <dbname> *help* returns a complete overview of all available command functions.

The database console can also be addressed remotely via the DBM server.

DB Console x_cons (II)

x_cons <dbname> <Command> [<interval>] [<repeat>]

<Command>):

	SHOW ACTIVE	[DW | SV | US | GC]
	SHOW ALL	
	SHOW AIO	(backup only)
show statistics/states	SHOW IO	
	SHOW DEV_IO	
show move info (load balancing)	SHOW MOVEINFO	
	SHOW QUEUES	
	SHOW REGIONS	
	SHOW RTE	
	SHOW RUNNABLE [DW | SV | US | GC]	
UKT sleep statistic	SHOW SLEEP	
	SHOW STATE	
	SHOW STORAGE	
suspend reasons	SHOW SUSPENDS	
show task counts	SHOW T_CNT	[DW | SV | US | T<taskindex>]
show tasks move info	SHOW T_MOVE	
show task queues	SHOW T_QUEUE	
show task regions	SHOW T_REG	
show task statistics	SHOW T_STAT	
	SHOW TASKS	
Thread time usage	SHOW THRD_TIMES	
	SHOW SLEEP	
	SHOW VERSIONS	
cancels the command of task	CANCEL <taskindex>	
displays help file	HELP	
time measurement	TIME <ENABLE | DISABLE>	
kills the session of task	KILL <taskindex>	

CANCEL	index	cancels the command executed by task <index>
KILL	index	kills the session of task <index>
SHOW [LONG | COMPRESS] (Unix)		
DEBUGLEV	level	set debug level for the kernel
DEBUGTASK	index	writes back trace of task to knldiag
RESET T_CNT REGIONS (ALL)	obj_cnt	resets counter about the following objects: IO incl. local counters of any task
ERRIOR	devno	forces error at next read to devno
ERRIOW	devno	forces error at next write to devno
TIME	enable	enables time measurements
! command		execute shell command (Unix only)
QUIT		exit console driver (Unix only)

x_cons Process Configuration (1)

```
x_cons <dbname> show rte
```

```
Special Threads:
 UNIX Threads
   tid name
 27696 COORDINATOR
 27702 CLOCK       Now 1146211614 = Fri Apr 28 10:06:54 2006
 27705 CONSOLE
 27711 TIMER       did 580100 sleeps. Now sleeping until 1146211616 for 5 s
 27712 REQUESTOR
 27713 DEV0

User Kernel Threads:
 UNIX Thread     Dispatch TaskSwitch Command  Active  Total   Task
   tid name        counter    counter counter   Tasks  Tasks   cluster
 27722 UKT1            2          0       0       1      1   TW
 27723 UKT2       219912          0       0       1      1   AL
 27724 UKT3            1          0       0       0      1   UT
 27725 UKT4        20082      16350       0      63     63   63*SV
 27726 UKT5        57759          3       0       1      3   1*GC,2*EV
 27727 UKT6       313711      21784       0      65     65   TI,64*DW
 27728 UKT7       839666       6170  673088      14     15   15*US
 27729 UKT8       398830      90454  309760      13     15   15*US
```

Kernel parameter (please do not change directly):

_TASKCLUSTER_01 **z.B. tw;al;ut;2000*sv,100*bup;10*ev,10*gc;**
_TASKCLUSTER_02 **z.B. ti,100*dw;15*us;**
_TASKCLUSTER_03 **z.B. equalize**

x_cons <dbname> show rte

This shows the distribution of the MaxDB threads among the operating system processes. The DB threads coordinator, console, timer, requestor and Dev0 each have their own operating system threads. The entire database kernel runs in a single process.

However, multiple database tasks (user task, log writer, utility task, and so on) can be located together in an operating system thread, which is called a UKT (user kernel thread). The MaxDB runtime environment uses internal tasking to administer these database tasks. Internal MaxDB administration takes up less operating system time, and gives you more control over the scheduling and prioritization of individual database sessions.

The database parameter MAXCPU is normally used to distribute the tasks automatically to the UKTs; the (support) database parameter TASKCLUSTER (requires change in the control file cserv.pcf) can also be used for this purpose, but only in consultation with SAP support.

x_cons Process Configuration (2)

`x_cons <dbname> show rte` (continued)

```
I/O via UKT and Device Processes:
  UNIX Thread Devspace                      Dev       Read      Write    Queue
   tid name   name                         index      count      count  len  max
 27723 UKT2   /sapdb/E70/saplog/DISKL001        5          0     114633   -- (--)
 27726 UKT5   /sapdb/E70/sapdata/DISKD0001      1        582         12   -- (--)
 27726 UKT5   /sapdb/E70/sapdata/DISKD0002      2        908         17   -- (--)
 27726 UKT5   /sapdb/E70/sapdata/DISKD0003      3        970         14   -- (--)
 27726 UKT5   /sapdb/E70/sapdata/DISKD0004      4        372          8   -- (--)
 27728 UKT7   /sapdb/E70/sapdata/DISKD0001      1       5235          1   -- (--)
 27728 UKT7   /sapdb/E70/sapdata/DISKD0002      2       5218          0   -- (--)
 27728 UKT7   /sapdb/E70/sapdata/DISKD0003      3       5162          0   -- (--)
 27728 UKT7   /sapdb/E70/sapdata/DISKD0004      4       5129          0   -- (--)
 27728 UKT7   /sapdb/E70/saplog/DISKL001        5          7          2   -- (--)
 27730 DEV    knltrace                          0          0          1    0 (1)
 27732 DEV    /sapdb/E70/sapdata/DISKD0001      1      14262       3242    0 (31)
 27731 DEV    /sapdb/E70/sapdata/DISKD0001      1         58       1229    0 (30)
 27734 DEV    /sapdb/E70/sapdata/DISKD0002      2      14135       3279    0 (30)
 27733 DEV    /sapdb/E70/sapdata/DISKD0002      2         36        983    0 (30)
 27736 DEV    /sapdb/E70/sapdata/DISKD0003      3      14023       3265    0 (32)
 27735 DEV    /sapdb/E70/sapdata/DISKD0003      3         56       1136    0 (32)
 27738 DEV    /sapdb/E70/sapdata/DISKD0004      4      14233       2977    0 (32)
 27737 DEV    /sapdb/E70/sapdata/DISKD0004      4         66       1104    0 (31)
 27740 DEV    /sapdb/E70/saplog/DISKL001        5          1         28    0 (1)
 27739 DEV    /sapdb/E70/saplog/DISKL001        5          0          0    0 (0)
```

Kernel parameter:

_IOPROCS_PER_DEV

Number of I/O threads per volume, see unit Kernel Parameters

Abbreviations of the Database Tasks in TASKCLUSTER:

Abbreviation

- tw Trace writer, writes kernel traces and dumps
- ti Task for timeout monitoring
- al Log writer
- dw Tasks for cache monitoring and asynchronous cache displacement as well as savepoint I/O
- ut Utility task for administration tasks (start backup, recovery, and so on).
- sv Server processes for backup I/O and special operations such as parallel index generation
- us User tasks for executing SQL statements
- gc Garbage collector
- ev Event task

x_cons Task Activity

```
x_cons <dbname> show active [<interval>] [<repeat>]
```

```
x_cons E70 show active 10 6

 ID    UKT UNIX  TASK       APPL Current       Timeout  Region        Wait
           tid   type        pid state         priority cnt try          item
T146    7    -1  User      28069 Running              0 220  99        741131(r)
T147    7    -1  User      28072*Runnable            48 0   111        741131(r)
T152    8    -1  User      28071*Runnable            56 0    76        424309(r)
T154    8    -1  User      28070 Running              0 55             424309(r)

T2      2     -1 Logwr        -1 IO Wait (W)          0 1     5          1978(s)
T152    8     -1 User      28069 LogIOwait(234)       0 0              424800(s)

T66     6     -1 Pager        -1 Vvectorio            0 0              3258(s)
T67     6     -1 Pager        -1 IO Wait (W)          0 0     1        3258(s)
T87     4     -1 Savepnt      -1 PagerWaitWritr       0 0              234617(s)

T75     4     -1 BUPvol       -1 AsynWaitRead         0 0              11368(s)
T76     4     -1 BUPvol       -1 AsynWaitWrite        0 0              11368(s)
T159    8     -1 User      28072 IO Wait (R)          0 0     2        429215(s)

T152    8     -1 User      28069 InvRootExcl          0 0 74573         24185(r)
T154    8     -1 User      28070 Running              0 55             438561(r)

T142    7     -1 User         0* Vwait                0 0              745843(s)
T157    8     -1 User         0* IO Wait (R)          0 0     1        852579(s)
```


x_cons <serverdb> show active [<interval>] [<repeat>]

Presents an overview of the states of all active tasks.

Appl pid

- Process ID of the application program linked to the task. An asterisk (*) before the PID indicates that the process ID is on a separate computer and is being accessed remotely.

Region

- cnt: Displays the number of times the region has been accessed since the task has been running.
- try: The number of the queried or held region

UKTsleep

- Number of semaphore waits per UKT

x_cons: Task States (1)

AsynClose	closes an I/O port after backup or recovery
Asyncntl	determines parameter or initialises a backup device
AsynIO	asynchronous I/O (during backup oder recovery)
AsynOpen	opens an I/O port for backup or recovery
AsynWaitRead	waits for an I/O operation to end, then read (backup or recovery)
AsynWaitWrite	waits for an I/O operation to end, then write (backup or recovery)
Command reply	delivers a result to the application
Command wait	task is waiting for a new request
Connect wait	task is free for a new session
DcomObjCalled	a DB-procedure or a COM-object is currently executed
Diaginit	initialises the datenbase internal trace files
Inactive	task is in initial state and has no resources yet
InsertEvent	creates an event
IO Wait (R)	waiting for I/O (R=read)
IO Wait (W)	waiting for I/O (W=write)
IO2 Wait (R)	
IO2 Wait (W)	
Locked	task is locked during kernel shutdown (to prevent rescheduling)
Mark for Start	
Net Cmd wait	

In a system with one CPU, only one task can be running at a given time. If x_cons nevertheless shows two tasks running, this is due to unprotected access.

x_cons: Task States (2)

Not Connected	
ReschedulеMsec	brief wait, continues automatically
Runnable	immediately runnable
Running	running, using CPU time
Stopped	suspended by kernel and waiting to proceed running
Terminated	task or datenbase session has been canceled
UNKNOWN	task state unknown
V2blockio	
V2info	
Vacknowledge	
Vattach	opens I/O ports (volumes, normal operation)
Vbegexcl	waiting for protected memory access
Vblockio	runnable after protected memory access
Vcopyvolume	
Vcreate	
Vdetach	closes I/O-ports (volumes, normaler operation)
Vdevsize	determines volume size or formats a volume
Vdualvectorio	performs a vector-I/O-operation on two volumes in parallel
Vendexcl	leaving a protected ares
VenterRWRegion	waiting to access a protected region (reader/writer region)

x_cons: Task States (3)

Vfclose	closes a file
Vfopen	opens a file
Vfread	reads a file
Vfwrite	writes a file
Vkill	a task has been canceled externally
VleaveRWRegion	leaves a protected region (reader/writer region)
Vnclear	
Vnclose	
Vnopen	
Vnrequest	
Vnrestart	
Vnshutdown	
Vopmsg	message written to files knldiag, knldiag.err and/or opmsg[n]
Vrelease	exits a database session
Vshutdown	changing database state from ONLINE to ADMIN
Vsleep	brief wait, continues automatically
Vsuspend	suspended and waiting to be explicitly activated by another task (e.g. for B*-Tree locks (very brief) or log I/O)
Vvectorio	performs a vector-I/O-operation (reading or writing)
Vwait	waiting to be explicitly activated by another task (e.g. waiting for an SQL-lock)
WaitForEvent	waiting for an event
Yielding	Briefly cedes control of CPUs during Busy Waiting

x_cons Task Detail

```
x_cons <dbname> time enable
x_cons <dbname> show t_c t<task_index>
```

```
-------------------  T25   User      ( pid = 23163  ) ----------------------
 remote_node   : myserver                              remote_pid    : 23163
 dispatcher-cnt: 127292                                command-cnt   : 30477
 total_excl-cnt: 9110558                               self_susp-cnt : 433
 dev_write_io  : 19         dev_write_pg  : 19         avg_dev_wr_tm : 0.0895
 state_vwait   : 11                                    avg_vwait_time: 4.1446
 state_vsusp   : 682                                   avg_vsusp_time: 0.0684
 rcv_rpl_count : 2296       rcv_rpl_long  : 46         avg_rcv_rpl_t : 0.1677
 rpl_rcv_count : 2296                                  avg_rpl_rcv_t : 0.0222
 dev_que_len_0 : 18         dev_que_len_1 : 0          dev_que_len>1 : 0
-------------------------------------------------------------------------------
```

#Statements > 1 second

Avg.. I/O time for Dev process

x_cons <serverdb> show t_c t<task_index> displays highly- detailed measurement values for individual database tasks. In this way, you can, for example, monitor the DB activity of an application while it remains connected to a database task (no permanent release/connect).

Furthermore, with *x_cons <serverdb> time enable* you can activate precise time measurement of the different database states. Depending on the operating system, this time measurement costs between 1% and 5% performance.

Much of the output of the '*show t_c*' function was developed exclusively for developers, however, some of the values are of more general interest in special situations.

dispatcher-cnt	Count of how often the task passed control to the UKT dispatcher, because it could not run, its time slot had expired, or another task was prioritized.
total_excl-cnt	Number of region accesses
command-cnt	Communication count between application and kernel
self_suspend-cnt	Number of task suspensions in which the task remained executable but still gave up control
<dev/self>_<read/write>_io	Number of I/Os via UKT (self) or DEV threads (dev)
<dev/self>_<read/write>_tm	Duration of an I/O via UKT (self) or DEV threads (dev)
state_vwait	Number of waits on SQL locks
avg_vwait_time	Average wait time for an SQL lock
avg_rcv_rpl_t	Average processing time of an SQL statement in the database kernel
rcv_rpl_long	Number of SQL statements with a processing time of more than one second

x_cons I/O Activities

```
x_cons <dbname> show io
```

Volume	No.	Read(s)	RPages	Write(s)	WPages
/sapdb/E70/sapdata/DISKD0001	1	10539	10539	11	12
/sapdb/E70/sapdata/DISKD0002	2	10525	10525	23	27
/sapdb/E70/sapdata/DISKD0003	3	10338	10338	22	22
/sapdb/E70/sapdata/DISKD0004	4	10000	10000	25	25
/sapdb/E70/saplog/DISKL001	5	0	0	36	36
total I/O		41402	41402	117	122

```
x_cons <dbname> show dev_io
```

I/O via Volume Thread:

UNIX tid	Volume name	Read count	avg_read time	Write count	avg_write time
19278	/sapdb/E70/sapdata/DISKD0001	15624	0.0079	5	0.0086
19277	/sapdb/E70/sapdata/DISKD0001	245	0.0201	6	0.0074
19280	/sapdb/E70/sapdata/DISKD0002	17028	0.0074	16	0.0009
19279	/sapdb/E70/sapdata/DISKD0002	383	0.0070	7	0.0031
19282	/sapdb/E70/sapdata/DISKD0003	15342	0.0074	18	0.0048
19281	/sapdb/E70/sapdata/DISKD0003	183	0.0075	4	0.0205
19284	/sapdb/E70/sapdata/DISKD0004	15767	0.0078	13	0.0002
19283	/sapdb/E70/sapdata/DISKD0004	269	0.0091	12	0.0003
19286	/sapdb/E70/saplog/DISKL001	0	0.0000	1	0.0002

The command *show io* displays the number of read and write operations per volume as well as the number of 8 KB pages read. These numbers are independent of whether the I/O are synchronous or asynschonous.

Show dev_io displays the number of read and write operations of the I/O threads. If time measurements are activated, the command also displays the average I/O times.

Performance tables: Process Configuration

SYSMON_SPECIAL_THREAD

- Shows special threads
- Similar to „x_cons <serverdb> show rte“ (see x_cons runtime configuration (2))

SYSMON_UKTHREAD

- Displays all threads containing tasks
- Analog to „x_cons <serverdb> show rte“
- (see x_cons runtime configuration (1))

SYSMON_IOTHREAD

- Shows I/O threads
- analog to „x_cons <serverdb> show rte“
- (see x_cons runtime configuration (2))

Performance Tables/ Database Console

Much of the data generated with x_cons is also accessible through tables. This this performance data can also be displayed by other tools (SQLStudio, SAP WebAS->DB50).

The columns of the respective tables largely correspond to those of the database console.

Performance tables: Task Activities (1)

SYSMON_TASK

- Shows all tasks
- analog to „x_cons <DBNAME> show tasks“

SYSMON_US

- shows all User Tasks
- analog to „x_cons <DBNAME> show tasks us“

SYSMON_DW

- shows all DataWriter Tasks
- analog to „x_cons <DBNAME> show tasks dw“

SYSMON_SV

- shows all Server Tasks
- analog to „x_cons <DBNAME> show tasks sv“

Performance tables: Task Activities (2)

SYSMON_ACTIVE_TASK / SYSMON_RUNNABLE

- shows all active tasks
- analog to „x_cons <serverdb> show [active|runnable]“

SYSMON_US_ACTIVE / SYSMON_US_RUNNABLE

- shows all active User Tasks
- analog to „x_cons <serverdb> show [active|runnable] us“

SYSMON_DW_ACTIVE / SYSMON_DW_RUNNABLE

- shows all active DataWriter Tasks
- analog to „x_cons <serverdb> show [active|runnable] dw“

SYSMON_SV_ACTIVE / SYSMON_SV_RUNNABLE

- shows all active Server Tasks
- analog to „x_cons <serverdb> show [active|runnable] sv“

Performance tables: I/O

SYSMON_BACKUPIOACCESS

- contains all volumes / backup-media, which have been used for a save / restore operation
- analog to „x_cons <serverdb> show aio“

SYSMON_IOACCESS

- all volumes, which have been accessed during MaxDB operation (but no SAVE / RESTORE)
- analog to „x_cons <serverdb> show io“

Performance tables: Miscellaneous

SYSMON_REGION

- all synchronized objects necessary to sync „critical areas“.
- analog to „x_cons <serverdb> show region“

SYSMON_STORAGE

- shows the tasks memory requirements
- analog to „x_cons <serverdb> show storage“

SYSMON_TASK_DETAIL

- detailed data for all [or a single] tasks
- analog to „x_cons <serverdb> show t_c [task_ID]“

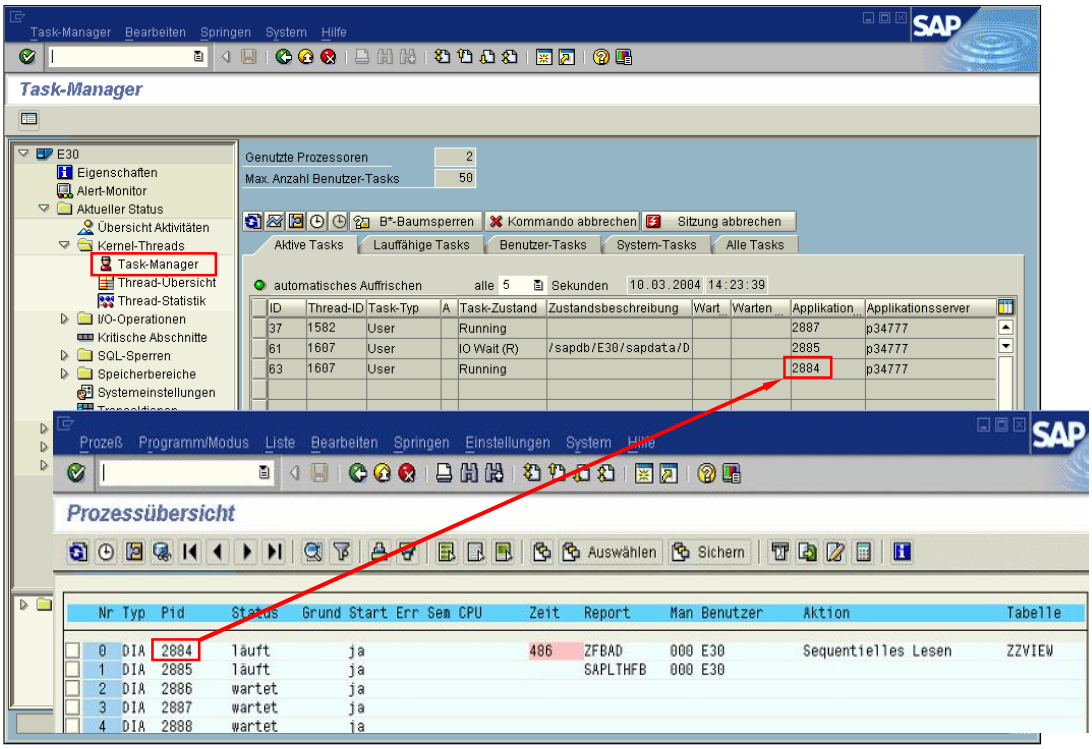
SYSMON_CONNECTION

- displays additional connect-Information e.g. the Vservers (x-server) PID when using remote communication
- analog to „x_cons <serverdb> show connection“

SYSMON_TOTALCOUNT

- sums all counters
- analog to „x_cons <serverdb> show total“

DB50: Kernel Threads – Task Manager (1)



Display of all database tasks and the current status. The system displays an overview of the database tasks and information about the current state of each individual task.

The following views are available: *Active Tasks, ExecutableTasks, User Tasks (task type User), System Tasks , All Tasks*.

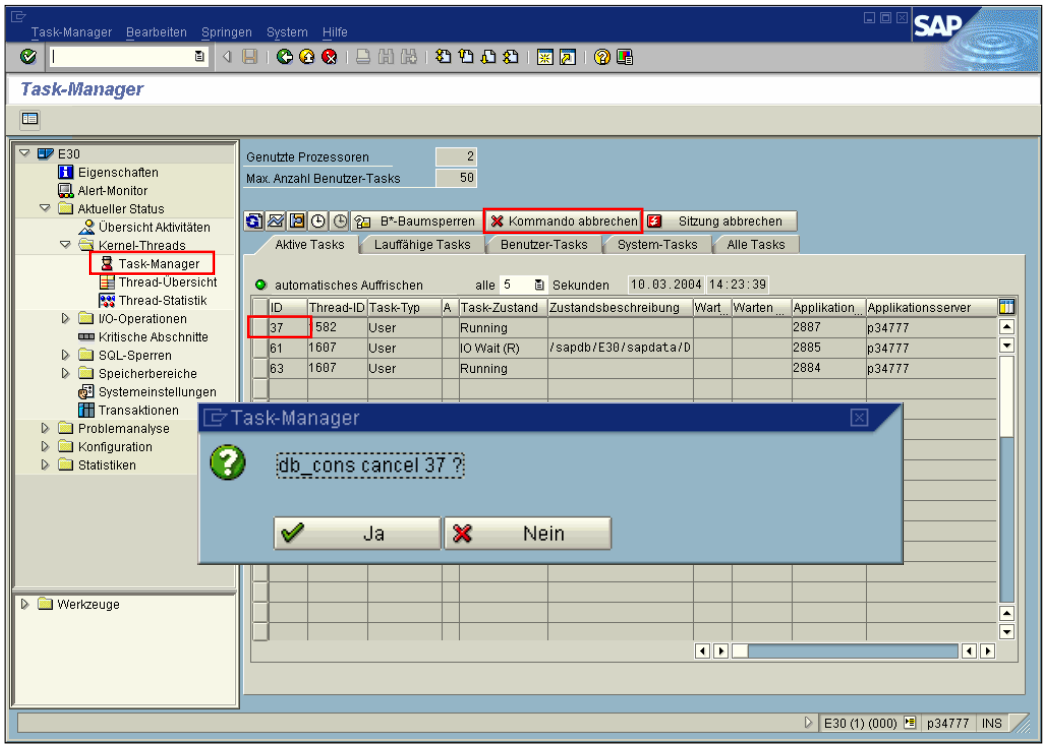
We use the task manager to analyze the following:

- For this database MAXCPU is set to 2. Thus the database can use 2 CPUs in parallel. Task T37 is running in another UKT (see Thread Overview, thread ID:1582) as task T61 and T63 (thread ID:1607). Tasks T37 and T63 can both have the *Running* status.
- We see a command (T61) that reads data from the disk to the cache - IO-WAIT (R).

In the Application column we see the process ID of the work process and via the Application Server column we see the SAP application server.

With transaction SM50, we can identify the application that caused the long-running command using the application PID (2887).

DB50: Kernel Threads – Task Manager (2)



With the task manager it is possible to terminate the respective task (T37) directly on the database level.

The information in the process overview can then be used to examine the application for possible programming errors or missing indexes.

Database Analyzer: a Typical Problem

Customer

… is reporting performance issues he thinks are database related

Support

… analyses the situation

- configuration? (caches, MAXCPU…)
- collisions? (SQL/BD locks, regions …)
- strategies? (used strategies, bad indices, current statistics.,…)
- I/O system? (log / data accesses?)
- …

… gathers data from system tables / x_cons

- tedious work
- time consuming

Database Analyzer: Objectives

Gathering relevant performance data with one tool

Replaces x_wizard / x_wiztrc

Flexible and upgradable through new rule sets

Release and instance independent

Remote access possible

CSN **note 530394** describes bottleneck analysis with the Database Analyzer.

The DBAnalyzer is available as of version 7.3.00 Build 25.

The tools ***x_wizard a***nd ***x_wiztrc*** were discontinued as of release 7.4.

Enhanceability

The tools ***x_wizard a***nd ***x_wiztrc*** were not configurable. The Executables had to be recreated for each platform every time an enhancement was carried out.

The logic and rules for monitoring with the ***Database Analyzer*** are defined by way of a **configuration file** (ASCII text). In case of changes or enhancements, you only have to cahnge the configuration file in the directory INSTROOT/env.

Release independence

As accesses to the system tables are defined in the **configuration file**, adjustments for new releases only require adjusting the configuration file. Consequently, this is release-independent, but the ***Database Analyzer*** itself is not. The configuration file takes account of the instance type (OTLP/LVC).

Remote capability

In addition to system tables, the ***x_wizard*** used output from “x_cons” that could only be generated on the database server.

The ***Database Analyzer*** uses only system tables. The data generated by “x_cons” can be queried via the SYSMON_…, system tables, which means they can be called remotely (e.g. via OSS).

Database Analyzer: Properties

Reporting weak spots in database configuration per given time intervals

Automatically classifies messages by color indicator

(info, light to severe performance problem)

Collecting monitor data each time interval

Database Analyzer

For routine monitoring of database operation in the production system, an interval of 15 minutes (-t900) is adequate. For short-term monitoring of database operation, a measuring interval of 10-30 seconds is recommended.

If class W3 warnings occur frequently, you should certainly try to remove the bottleneck. W3 warnings generally indicate that the runtime behavior of the database is severely compromised. If you suspect poor search strategies (rows read/rows qualified), a more precise analysis is unavoidable. The command monitor (DIAGNOSE MONITOR) is available for this purpose.

Not all *Database Analyzer* outputs are necessarily caused by actual bottlenecks. For example table scans can be useful in certain situations, long runtimes of statements can automatically occur with large datasets etc.

Database Analyzer: Technical Details

Executable dbanalyzer

- collects, assesses and stores data
- has (almost) no hard coded knowledge about system tables
- only rule based infrastruktur

Configuration file dbanalyzer.cfg

- INSTROOT/env/dbanalyzer.cfg

All changes concerning rules and assessments can be made in the configuration file without need to touch the softwares executable.

Configuration File: dbanalyzer.cfg

- Describes ("where and why") the data to be collected or calculated (***parameters***). These ***parameter*** are either taken from the database (system tables) or calculated from the ***parameter*** taken from the database. As the manual evaluation of parameters is time-consuming, the Database Analyzer formats the logged data.
- Describes the evaluation rules (**monitors**) for the parameters. The ***monitors*** have up to four conditions (***Information*** and ***Warnings 1*** through ***3***) and are logged in a way that takes account of the conditions. For logging the monitors, in the configuration file you can store a verbal assessment or even concrete instructions for the user.

Database Analyzer: Monitoring System Tables

e.g monitor for “DC_Hit” of /sapdb/programs/env/dbanalyzer75.cfg:

```
[Monitor]
ID              =       DC_HIT
Label           =       "Data cache hitrate (SQL Pages) " + DC_Hit + "%, " + DC_Fails + " of "+ DC_Acc + " accesses
failed"
Class           =       Caches
Description     =       For a running database application the data cache hitrate \
                        should not be less than 99%, otherwise too much data has \
                        to be read physically. Data cache hitrates less than 99% \
                        for intervals of 15 minutes or more must be avoided.
Warning3        =       DC_Hit < 96 \
                        && ( PReads ) > MAX_IDLE_IO_ALL_DEVS
Warning2        =       DC_Hit < 98 \&& ( PReads ) > MAX_IDLE_IO_ALL_DEVS
Warning1        =       DC_Hit < 99 \&& ( PReads ) > MAX_IDLE_IO_ALL_DEVS
Information     =       DC_Hit < 99 \ && ( PReads ) < MAX_IDLE_IO_ALL_DEVS
UserAction      =       In addition to enlarging the data cache (note the paging risk of the operating system), search for the
                        cause of the high read activity. Frequently, individual SQL statements cause...
```


Up to four conditions for triggering the monitor. Conditions are boolean expressions that refer to *parameters*.

The top-level message is stored in the *label.* The *label* is an expression that is calculated when the *monitor* is activated. This enables references to the *parameters.*

User-selected texts for *Description* and *UserAction.*

Database Analyzer: Features (1)

general warnings on

- low cache hitrates (data-/catalog-cache)
- high I/O rate
- low hitrates on Selects, Updates und Deletes (ratio found/read rows; optimizer strategy)
- log queue filling level too high / overflows
- lock list escalations

Database Analyzer: Features (2)

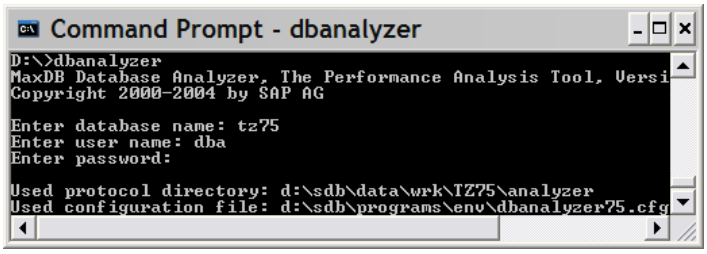
task specific warnings on

- poor I/O-times
- high lock waits (vwait/vsuspend)
- long command runtimes (receive/reply)
- high read activity (reads)
- a Usertask blockades in a certain state (e.g. Vwait, Vbegexcl…)

Database Analyzer: Program Start

calling the Database Analyzer

- from a UNIX- or DOS-Shell
  - start: dbanalyzer
  - -n <server>
  - -d <database>
  - -u <user,pwd>
  - -f <configfile>
  - -t <interval>,<number>
  - -o <outputdir> -c <level>
  - stop: dbanalyzer
  - -n <server> -d <database> -u <user,pwd> -f <configfile> -o <outputdir> -stop
- with the DBMCLI command dban_start



- per WebAS
  - manually via DB50 -> problem analysis -> DB bottlenecks
  - implicit start with SAP WebAS 6.20 Basis SP 37
- using the SAP CSS Support connection (SAP DB Connection → SAPDBCON)
  - Enables SAP support to collect and store data on a host of their choice

You can also call the Database Analyzers with the DBMCLI command dban_start . The Database Analayzer is then implicitly started in the background. The Database Analyzer call can be supplemented with various options.

-n <server>
Name of the computer on which the database instance is running. If you enter this argument, you have to specify a directory for logging with the -o switch.

-d <database>
Name of the database instance that is to be examined.

-u <user,pwd>
User name and password for authorization on the database server.

-f <configfile>
Indicates the name of the configuration file to be used. The standard setting specifies the file **dbanalyzer.cfg** in the directory **$INSTROOT/env**.

-t <interval>,<number>
Defines the time interval (in seconds) between two evaluations. If <number> is specified, the Database Analyzer ends automatically when it has reached the specified number.

-o <outputdir>
Specifies the directory in which the log files of the Database Analyzer are written. If you specify -n <server> at the time of the call, you also have to specify a log directory. If you fail to specify a log directory, logging is done in the **RUNDIRECTORY** of the database instance in the subdirectory **analyzer**.

-c <outputlevel>
Specifies that Database Analyzer output also be written to the console. In the standard setting, no output is written to the console. With <outputlevel> you can specify how much is to be output. The possible values are **1, 2, 3** and **4**.

-i Deletes (initializes) any pre-existing log files. This enables the logging of data from different databases in the same directory, which is otherwise prohibited. The data of the previously analyzed database are deleted in the process.

Database Analyzer: Time Interval

short term analysis: -t 10

- time interval 10 seconds
- evaluating data online

long term analysis: -t 900 (default)

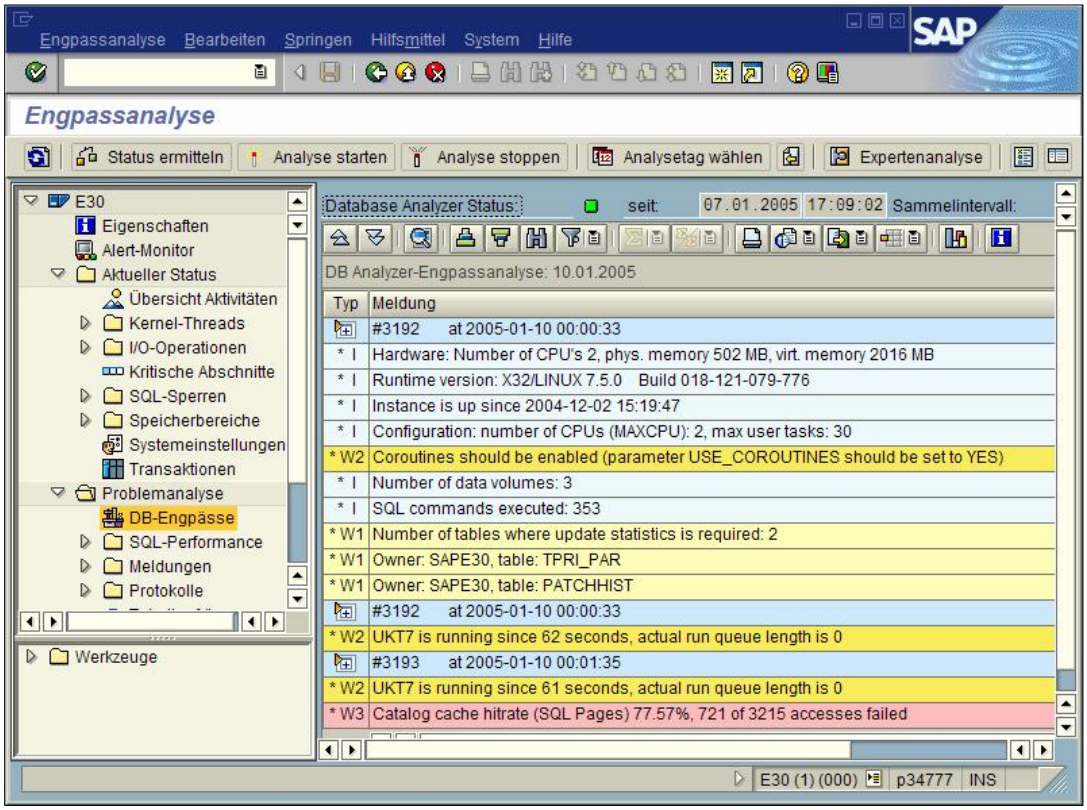
- time interval 15 Minuten
- If necessary start with “nohup Database Analyzer... &” and option -s in background (nur UNIX)
- All time data saved (ca. 1MByte/day)

in both cases

- creating and saving the protocol files

For routine monitoring of database operation in the production system, an interval of 15 minutes (-t900) is adequate. Logging should be activated with -p to obtain a retrospective overview of DB activities. For short-term monitoring of database operation, a measuring interval of 10-30 seconds is recommended.

Database Analyzer: Bottleneck Analysis (1)



As of support packages 6.20 SP37, the Database Analyzer starts automatically when the SAP WebAS system is started.

You can call the Database Analyzer from transaction DB50 -> Problem Analysis-> Bottlenecks. You can also stop and restart it from there.

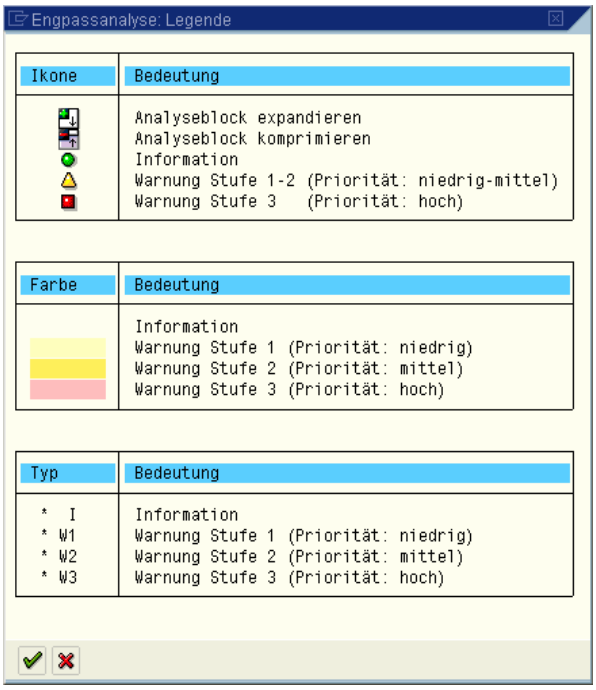
The default time interval for determining measurement data is 15 minutes. You can override this configuration stopping and restarting the Database Analyzer.

Each time the Database Analyzer is started, information about the configuration and performance-relevant data from system tables is output, including, for example, the number of tables that require an Update Statistics. You can determine the table names with a Select on the system table *sysupdstatwanted.*

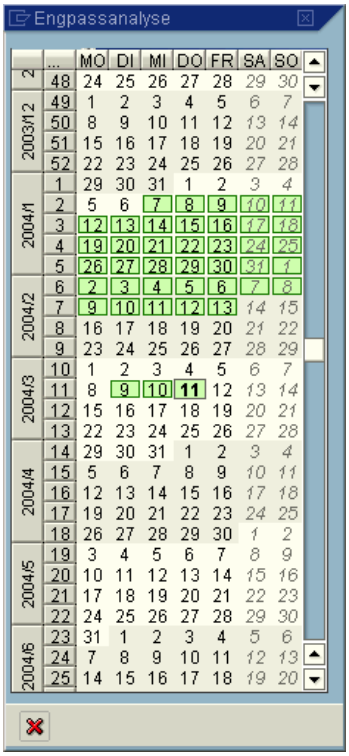
Detected bottlenecks are output in text form to rapidly provide database administrators with an overview of the possible causes of performance problems. The analysis can be performed just once or at regular intervals.

Database Analyzer: Bottleneck Analysis (2)

key



choosing analysis day

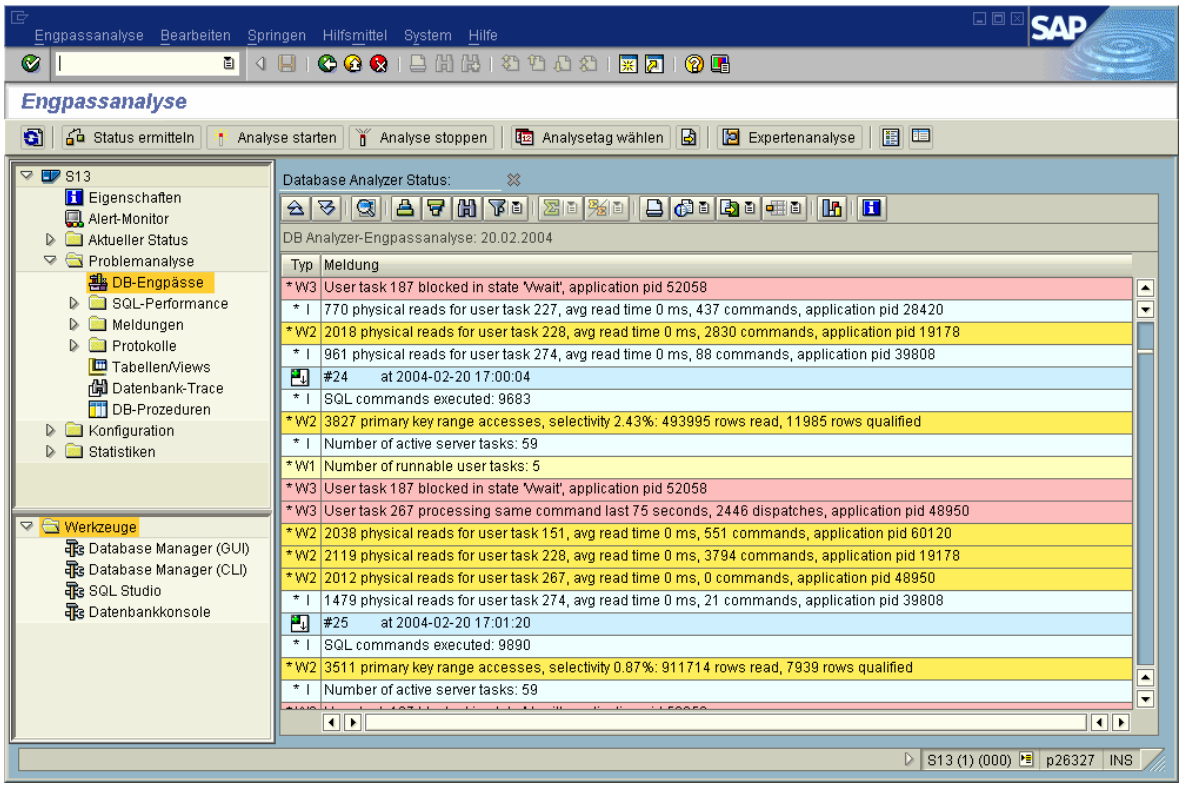


With the *Choose analysis day*, you can use the calendar to branch into the logs for a particular day. The days for which analysis data is available have a green background.

Logs are implicitly deleted periodically via the program *RSDBANCONTROL*. You can configure how long logs are kept using transaction DB59 in the integration data for the respective system. (6.20 as of basis SP 37).

Display color legend returns information on the icons, colors and types of messages used in the Database Analyzer log.

Database Analyzer: Bottleneck Analysis (3)



Excerpt from a Database Analyzer log from a performance analysis of a customer system.

Database Analyzer: Data Cache

low data cache hitrate:

<percentage> % <number> accesses, <number> successful, <count> unsuccessful

Cause:

- data cache too small
- SQL statements creating a lot of page reads (unselective commands, missing indices)

Action:

- Finding cause, e.g. with the Diagnose Monitor and pay attention to further Database Analyzer messages
- If nothing indicates an application or design problem: increase cache size to reduce risk of I/O sequentialization

Database Analyzer: Data Cache

Low data cache hit rate : <percentage> %

<number of> accesses, <number> successful, <number> unsuccessful

Explanations

The hit rate is too low when accessing the database cache. The data cache hit rate for a running database application should not be less than 98%; otherwise, too much data has to be read physically. For a short time, lower hit rates may occur; e.g., when reading tables for the first time, or when the table does not fit into 10% of the data cache with repeated table scans (only with LRU_FOR_SCAN = NO). Data cache hit rates under 98% for intervals of 15 minutes or more must be avoided.

User response

In addition to enlarging the data cache (note the paging risk in the operating system), search for the cause of the high read activity. Frequently, individual SQL statements cause a high percentage of the total logical and physical read activities. Enlarging the cache only transfers the load from the disk to the CPU although an additional index, for example, could transform a read-intensive table scan into a cheap direct access.

Database Analyzer: Paging Cache Entries

User task physical writes <number of phys. writes>

Causes:

- write transactions changing data pages in the cache
- data cache full, no more space for new pages
- before reading a new page, an already modified page has to be displaced

Action:

- increase cache size
- activate pager

Database Analyzer: cache displacements

Cache displacements: <number of> pages/second

Explanations

Modified pages are displaced from the data cache to disk because the data used by the applications cannot be completely kept in the data cache. If the size of the data cache were sufficient, the physical write would be delayed until the next SAVEPOINT and then be done asynchronously. Cache displacements result in synchronous I/O and should be avoided, if possible.

User response

Enlargement of the data cache. Particularly with larger data imports, the so-called *pagers* should be activated for regular asynchronous buffer flushes between the SAVEPOINTS database parameter _DW_IO_AREA_SIZE, _DW_IO_AREA_FLUSH, _DW_LRU_TAIL_FLUSH).

Database Analyzer: Selectivity

low access hitrates via <Optimizer Strategy>:

<percentage> % <number> accesses, <number> rows read, <number> rows qualified

Causes:

- disadvantageous execution of SQL commands. Too many reads necessary to fetch just a few results
- unfavourable SQL syntax/statement
- missing indices

Action:

- update statistics
- Find the responsible SQL commands with the help of Diagnose Monitor, analyse them and - if necessary - rewrite SQL or create index

Database Analyzer: selectivity

Explanations

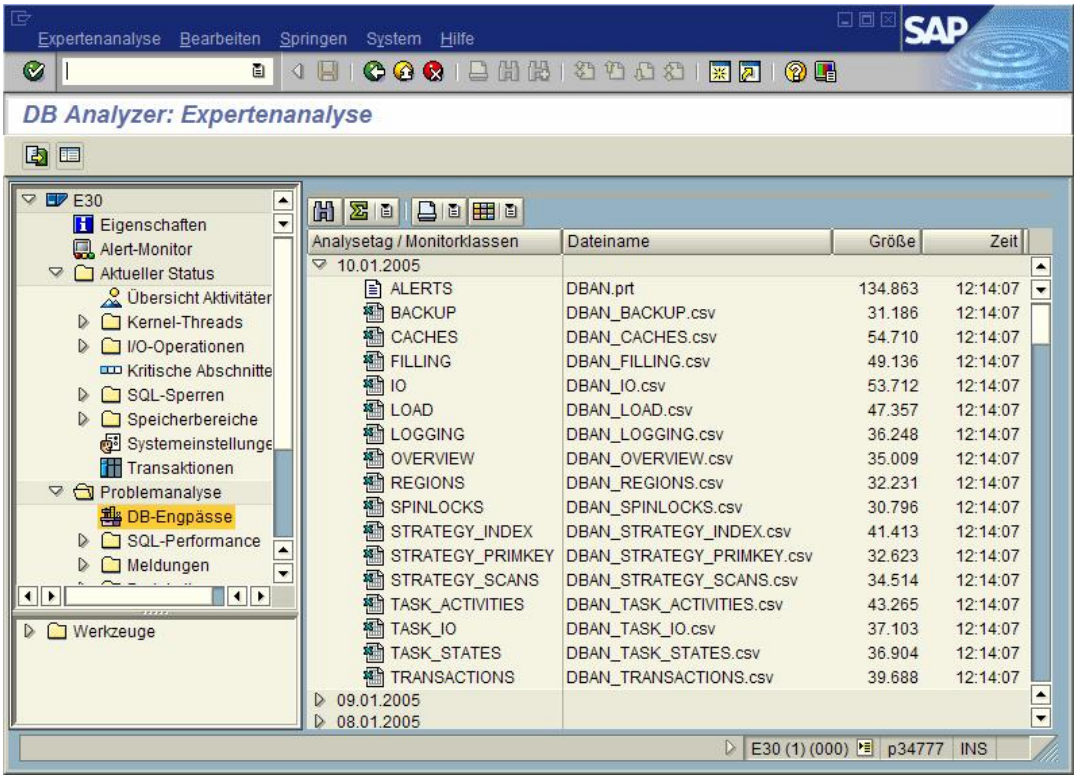
The relationship between read and found (qualified) rows is poor for a certain access strategy applied by the MaxDB Optimizer. This indicates a poor search strategy, caused either by the application (missing or insufficient indexes) or by poor formulation of SQL statements. Searching large quantities of data can seriously compromise the performance of the system as a whole due to the numerous negative effects (I/O, CPU load, etc.).

User response

First of all, see if MaxDB Optimizer is able to find a more suitable strategy after updating the internal database statistics. The update should be done directly from the SAP system with transaction DB13.

If this does not produce the desired result, search for the statement that triggers the unfavorable search strategy. The easiest way to do this is with DIAGNOSE MONITOR.

Database Analyzer: Expert Analysis

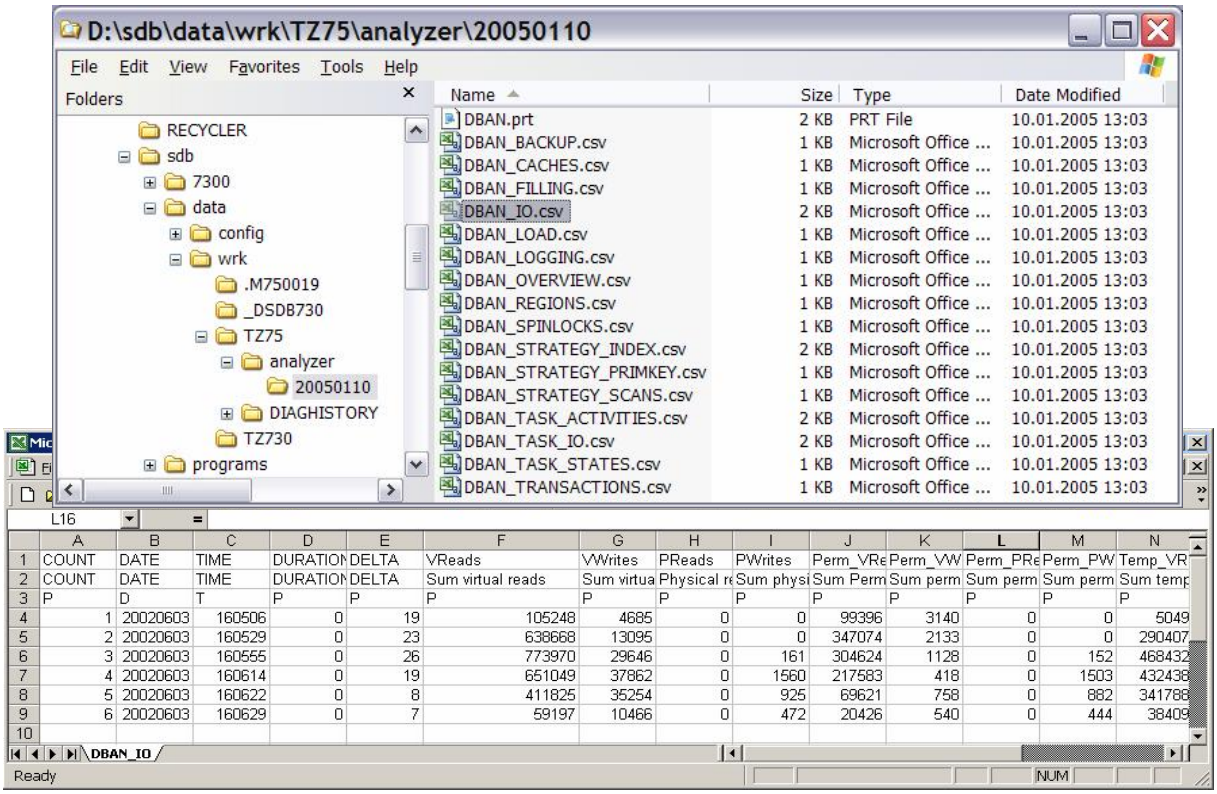


For detailed information on the monitor classes, choose *Expert Analysis.*

You get an overview of all available logs generated by the Database Analyzer. The significance of the individual logs is described on the following slides

To display the content of a file, double-click the relevant monitor class.

Database Analyzer: Log Files (1)



Storing performance data in the logs is useful when checking runtime behavior later.

The collected data is stored as "csv" files in the directory/YYYYMMDD specified with "-o".

If you start the Database Analyzer on the DB server, you can omit the "-o" entry.
In that case, logging is done in the run directory/YYYYMMDD

A directory contains the data from one day.

The data is grouped by contents and stored in different files. You can display the day in a table with MS Excel and from the WebAS.

Database Analyzer: Log Files (2)

DBAN.prt

- quick overview; records monitor data including all rule based values

DBAN_BACKUP.csc

- physical reads/writes for backup, read/write time (ms) for backup

DBAN_CACHES.csv

- accesses, successful, failed and hit rates of all caches (DATA, CATALOG,…)

DBAN_FILLING.csv

- database filling level (size, permanently/temporarily occupied…)

DBAN_IO.csv

- virtual/physical reads/writes (common, permanent, temporary, long)

DBAN_LOAD.csv

- accesses / selektivity of selects and fetches, inserts, updates, deletes

Database Analyzer: Log Files (3)

DBAN_LOGGING.csv

- number of actual log writes, log queue overflows, max log queue used

DBAN_OVERVIEW.csv

- summarizing the other protocols key points

DBAN_REGIONS.csv

- Region accesses, collisions, waits and dispatches

DBAN_SPINLOCKS

- spinlock collisions, read/write locks

DBAN_STRATEGY_INDEX.csv

- accesses / selectivity of index, index ranges and isolated index / index ranges

DBAN_STRATEGY_PRIMKEY.csv

- accesses / selectivity of primary key and primary key ranges

Database Analyzer: Log Files (4)

DBAN_STRATEGY_SCANS.csv

- accesses / selectivity of table and isolated index scans

DBAN_TASK_ACTIVITIES.csv

- SQL commands, task statistics (active, running, runnable…)

DBAN_TASK_IO.csv

- I/O number / duration for logwriter, user und datawriter Tasks

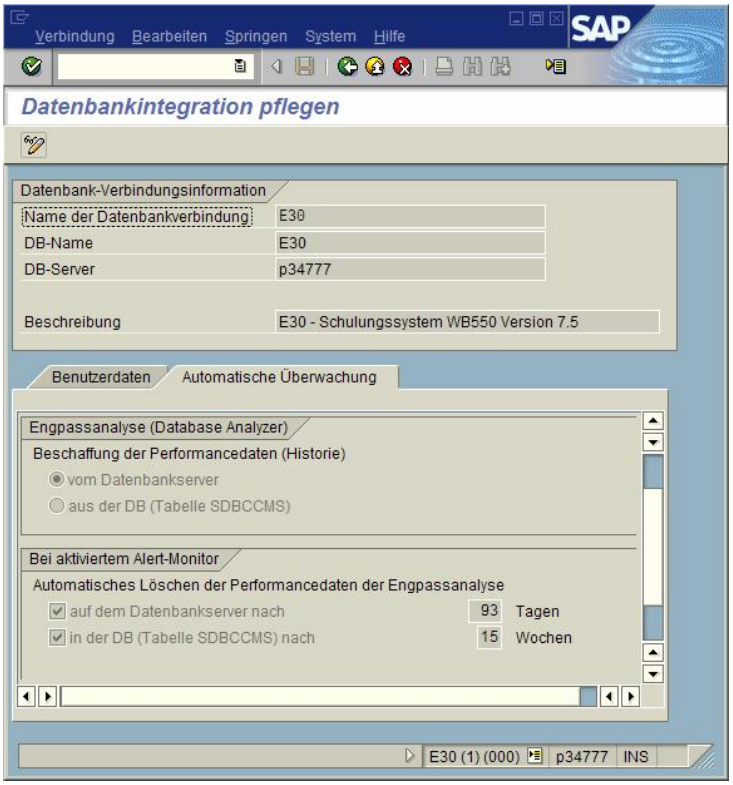
DBAN_TASK_STATES.csv

- number and elapsed time of processed commands
- number and used time in task states Vsuspend, Vwait, Vsleep

DBAN_TRANSACTIONS.csv

- number commands, prepares, executes, commits, rollbacks, subtrans, lock requst timeouts and lock request escalations

Database Analyzer: Log Files (5)



Via transaction DB59 -> *Integration Data*-> *Automatic Monitoring,* you can define the time interval at which Database Analyzer logs are deleted.

By default, the logs are stored for 93 days.

The corresponding information in the database table SDBCCMS, however, is kept for 15 weeks. For more information, see **note 530394**.

You can make your own personal settings by choosing Display/Change.

Command Monitor (Diagnose Monitor)

Automatic tracking of problematic commands

criteria:

- page accesses
  - registers SQL commands exceeding a set limit for page accesses
- runtime
  - records SQL commands exceeding a supplied runtime (in seconds)
- selectivity
  - Tracks SQL commands falling below a given ratio of qualified and read records

THE tool for identifying long running SQL commands

Logging can be switched on directly in SAP WebAS using transaction DB50 or with the following dbmcli command:

dbmcli -n <dbserver> -d <dbname> -u ... -uSQL sapr3,sap sql_execute

diagnose monitor selectivity <number>
| read <number>
| time <number>
| rowno <number>
| off

Selectivity:	Ratio of qualified to total records read is below the specified threshold value
Read (page accesses):	Specified number of virtual reads exceeded
Time (runtime):	Runtime of the command exceeds the specified time in milliseconds
Rowno: 255)	Number of statements to be stored as specified in SYSMONITOR (default
Off:	Deactivation

Performance Tables: Command Monitor

Commands and measured data are stored in tables SYSPARSEID, SYSMONITOR and SYSMONDATA:

SYSPARSEID

- stores Parse-ID and SQL command string

SYSMONITOR

- Periodically overwritten, max. 3000 commands
- contains Parse-ID and performance data

SYSMONDATA

- stores command data

Performance tables

The tables SYSMONITOR and SYSPARSEID are filled following activation of the DIAGNOSE MONITOR. SYSPARSEID can grow to any size, SYSMONITOR is overwritten cyclically.

The table SYSPARSEID contains the parse ID PARSEID for every parsed statement and the command string SQL_STATEMENT.

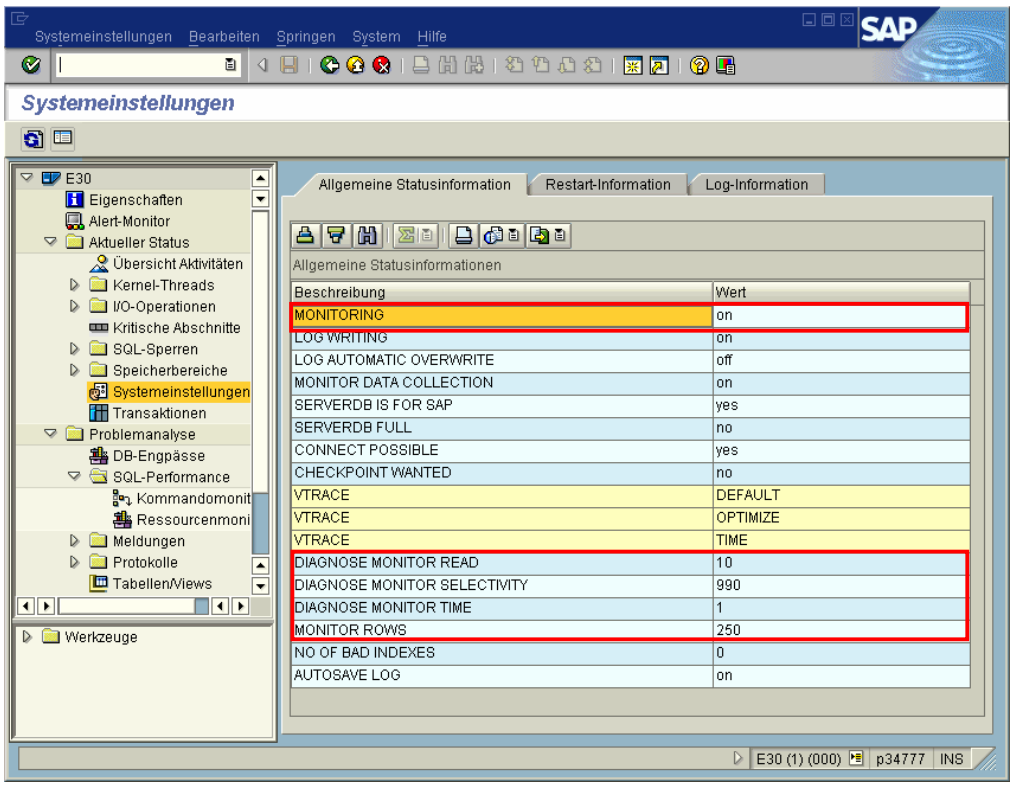
COLUMN NAME		MOD	DATA TYPE	CODE	LEN
PARSEID		KEY	CHAR	BYTE	12
LINKAGE		KEY	FIXED		1
SELECT_PARSEID	OPT	CHAR	BYTE		12
OWNER		OPT	CHAR	ASCII	18
SQL_STATEMENT	OPT	CHAR	ASCII		3900

Performance Tables: Example

Tables are joined for analysis

```
SELECT rows_read, rows_qual, strategy, runtime,
physical_io, sql_statement, substr(data,1,25)
FROM  sysmonitor, sysparseid, sysmondata
WHERE sysmonitor.parseid = sysparseid.parseid
AND   sysmonitor.sysk    = sysmondata.sysk
ORDER BY runtime DESC
```


DB50: System Settings



When you do a performance analysis, check the system settings to ensure that the performance analysis tools are working.

Monitoring: If monitoring is active, general performance-relevant data is stored in the system tables. Monitoring is automatically activated by the start scripts when you start the SAP WebAS and the database.

Monitor Data Collection: activates logging of transfer values in the Where condition of each SQL statement in the command monitor.

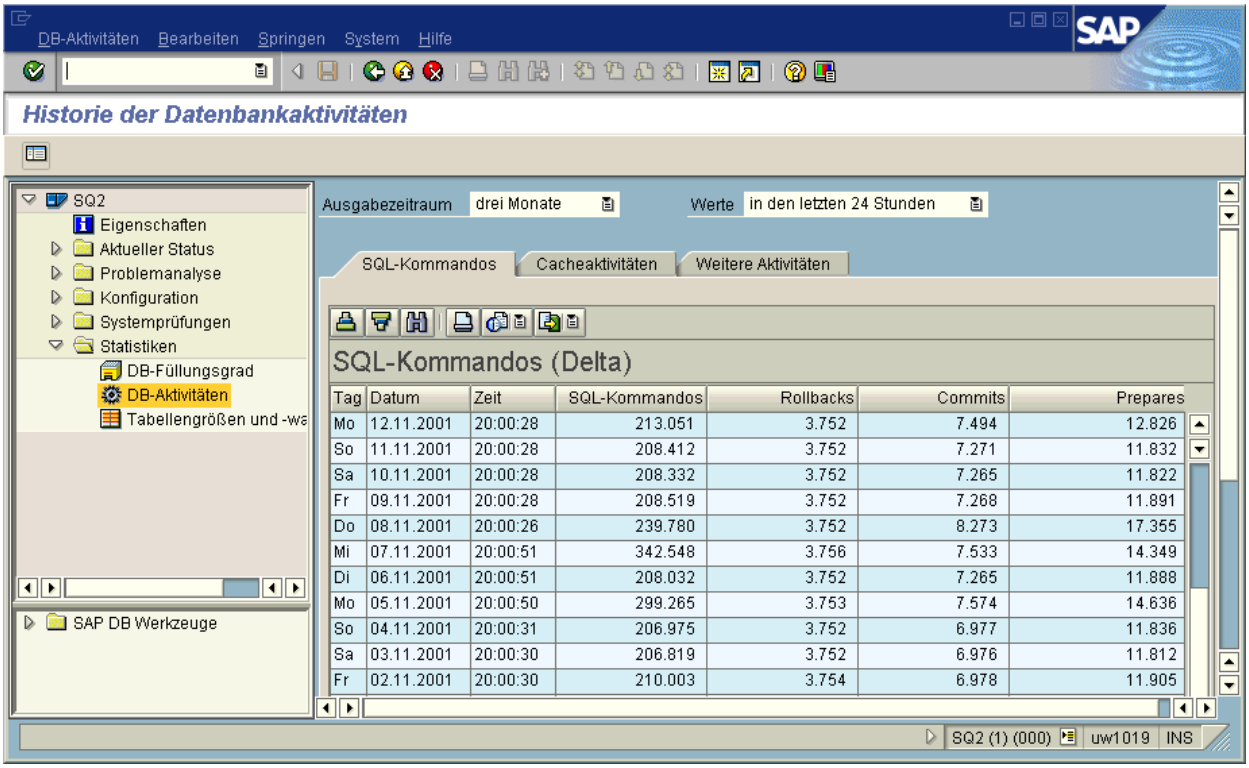
Diagnose Monitor Read / Selectivity: display the settings with which the command monitor (DIAGNOSE MONITOR) was started. If the command monitor is not active, the corresponding entries in the system settings are missing.

Diagnose Analyze and Diagnose Analyze Filter: Ressource monitor (see below)

Monitor Rows: Number of SQL statements

No of bad Indexes: indicates if there are defective indexes in the database. If an index is defective, it cannot be used for access.

DB50: Statistics – DB Activity (1)



Tag	Datum	Zeit	SQL-Kommandos	Rollbacks	Commits	Prepares
Mo	12.11.2001	20:00:28	213.051	3.752	7.494	12.826
So	11.11.2001	20:00:28	208.412	3.752	7.271	11.832
Sa	10.11.2001	20:00:28	208.332	3.752	7.265	11.822
Fr	09.11.2001	20:00:28	208.519	3.752	7.268	11.891
Do	08.11.2001	20:00:26	239.780	3.752	8.273	17.355
Mi	07.11.2001	20:00:51	342.548	3.756	7.533	14.349
Di	06.11.2001	20:00:51	208.032	3.752	7.265	11.888
Mo	05.11.2001	20:00:50	299.265	3.753	7.574	14.636
So	04.11.2001	20:00:31	206.975	3.752	6.977	11.836
Sa	03.11.2001	20:00:30	206.819	3.752	6.976	11.812
Fr	02.11.2001	20:00:30	210.003	3.754	6.978	11.905

You start the detailed performance analysis by checking the statistics to see if there is anything unusual in the functioning of the system today.

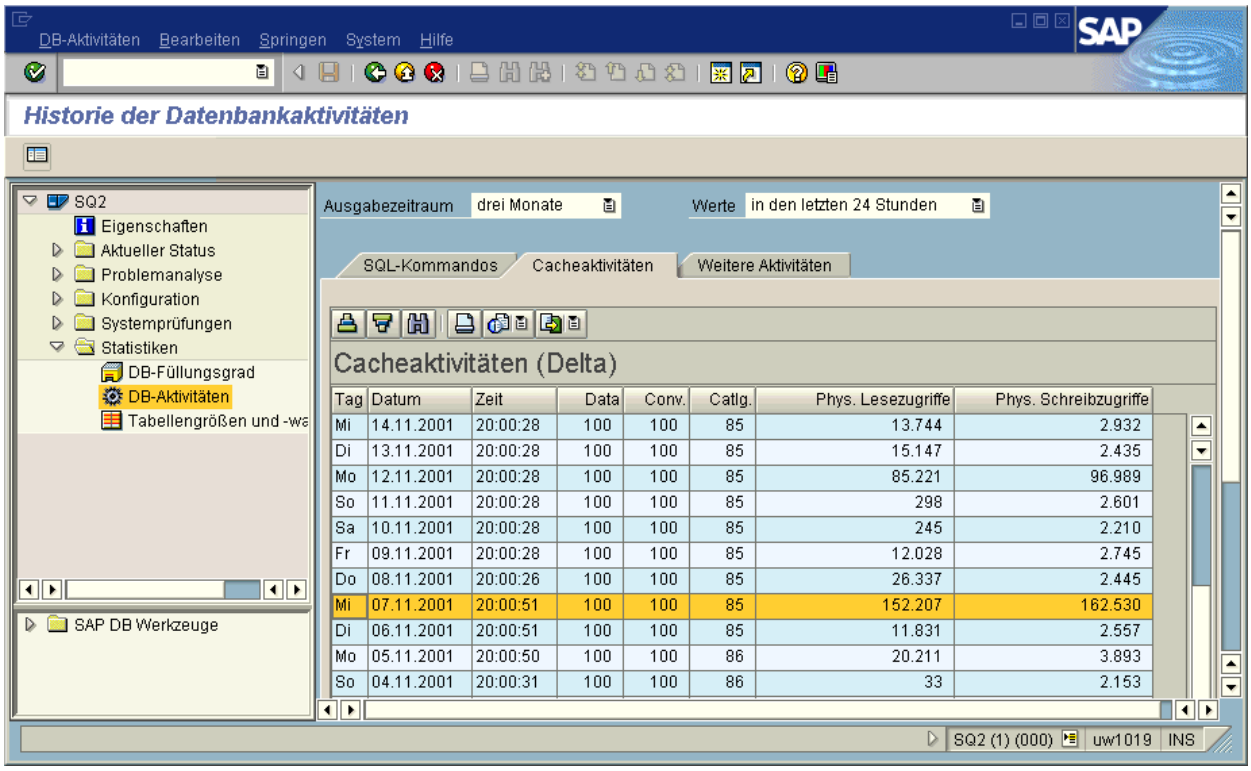
Display the database activities for a certain point in time. Let's have a look at Wednesday, 7 November, 2001.

The overview of database activities is logged daily by a collector (COLLECTOR_FOR_PERFORMANCEMONITOR).

SQL Commands: Total number of SQL statements, commits, rollbacks, prepares

We see that on Wednesday, 7 November, a somewhat higher number of SQL statements was executed than on other days.

DB50: Statistics – DB Activity (2)

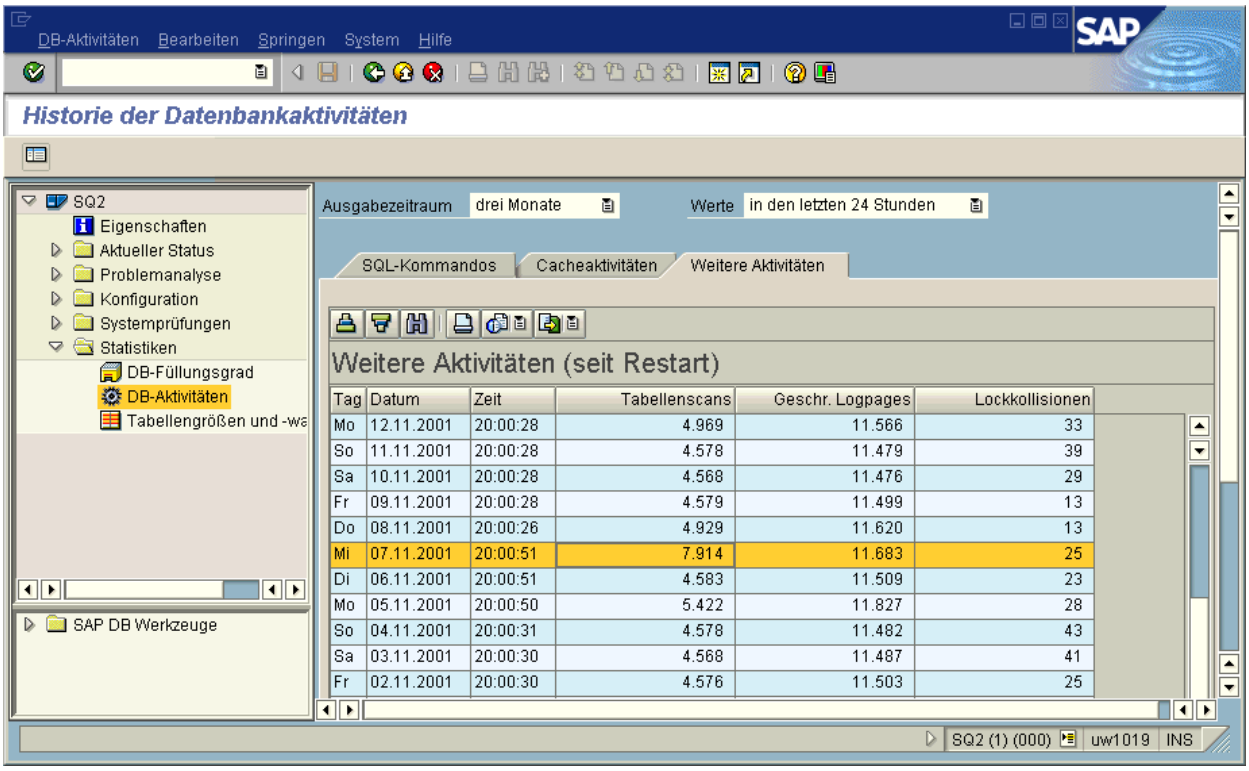


Tag	Datum	Zeit	Data	Conv.	Catlg.	Phys. Lesezugriffe	Phys. Schreibzugriffe
Mi	14.11.2001	20:00:28	100	100	85	13.744	2.932
Di	13.11.2001	20:00:28	100	100	85	15.147	2.435
Mo	12.11.2001	20:00:28	100	100	85	85.221	96.989
So	11.11.2001	20:00:28	100	100	85	298	2.601
Sa	10.11.2001	20:00:28	100	100	85	245	2.210
Fr	09.11.2001	20:00:28	100	100	85	12.028	2.745
Do	08.11.2001	20:00:26	100	100	85	26.337	2.445
Mi	07.11.2001	20:00:51	100	100	85	152.207	162.530
Di	06.11.2001	20:00:51	100	100	85	11.831	2.557
Mo	05.11.2001	20:00:50	100	100	86	20.211	3.893
So	04.11.2001	20:00:31	100	100	86	33	2.153

Cache activities: Cache hit rates, number of reads and writes to the database

The cache statistics for 7 November show that the cache hit rates were good, but the number of physical read and write accesses was significantly higher than on the other days.

DB50: Statistics – DB Activity (3)

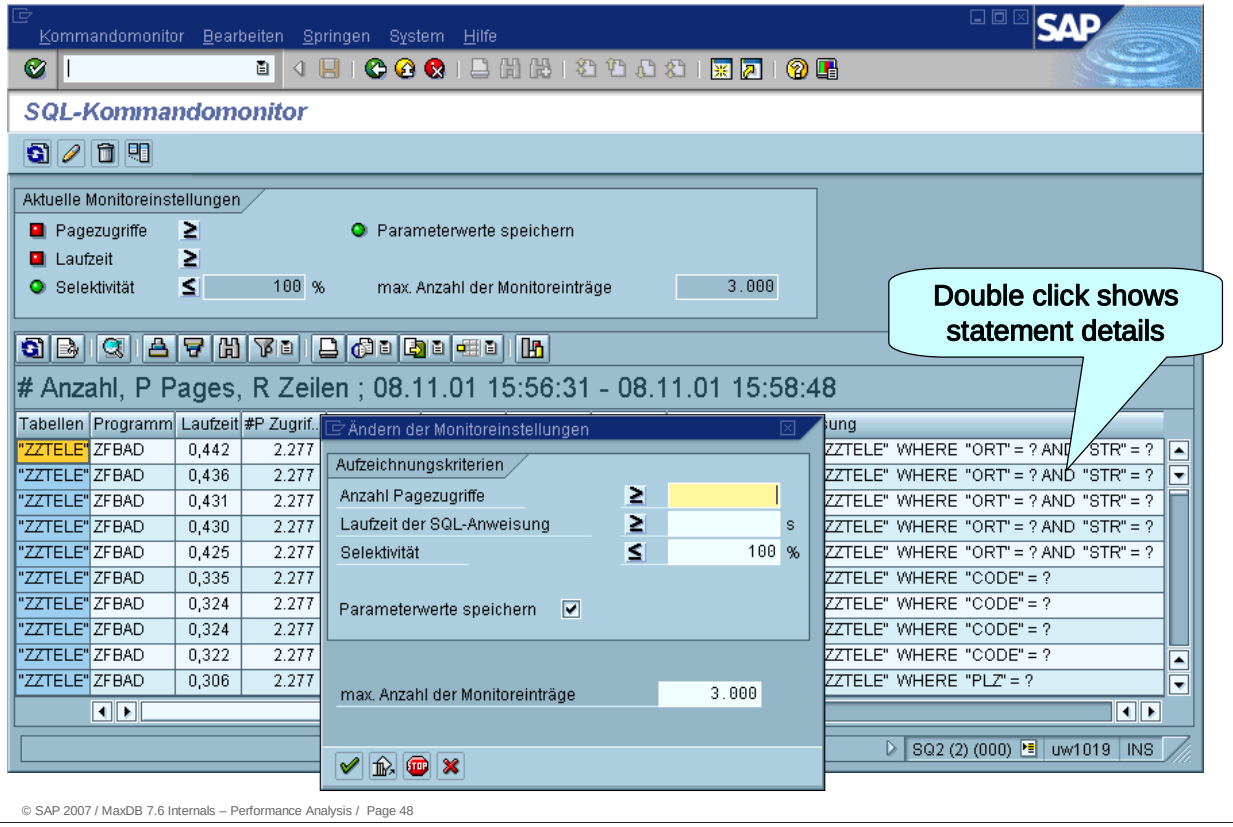


Tag	Datum	Zeit	Tabellenscans	Geschr. Logpages	Lockkollisionen
Mo	12.11.2001	20:00:28	4.969	11.566	33
So	11.11.2001	20:00:28	4.578	11.479	39
Sa	10.11.2001	20:00:28	4.568	11.476	29
Fr	09.11.2001	20:00:28	4.579	11.499	13
Do	08.11.2001	20:00:26	4.929	11.620	13
Mi	07.11.2001	20:00:51	7.914	11.683	25
Di	06.11.2001	20:00:51	4.583	11.509	23
Mo	05.11.2001	20:00:50	5.422	11.827	28
So	04.11.2001	20:00:31	4.578	11.482	43
Sa	03.11.2001	20:00:30	4.568	11.487	41
Fr	02.11.2001	20:00:30	4.576	11.503	25

Other Activities: Number of table scans, log activities, lock information

On 7 November, the number of table scans is notable. It, too, is very high compared to the other days. This all implies an application that is "problematic" in performance terms.

DB50: Problem Analysis – Command Monitor (1)



The command monitor (Diagnose Monitor) allows you to identify long-running SQL statements. This tool is intended for short analyses, since the number of recorded SQL statements is limited. Specify criteria to restrict the volume and type of SQL statements that are recorded.

Within SAP WebAS, logging is activated via transaction DB50 ? Problem Analysis? SQL Performance ? Command Monitor. You can set up various criteria in succession. A statement is logged when at least one of the criteria is fulfilled.

Choose *Command Monitor –> Change Monitor Settings* to determine the recording criteria by which SQL statements are logged in the command monitor tables.

Number of page accesses: An SQL statement is logged when the number of page accesses exceeds the specified value.

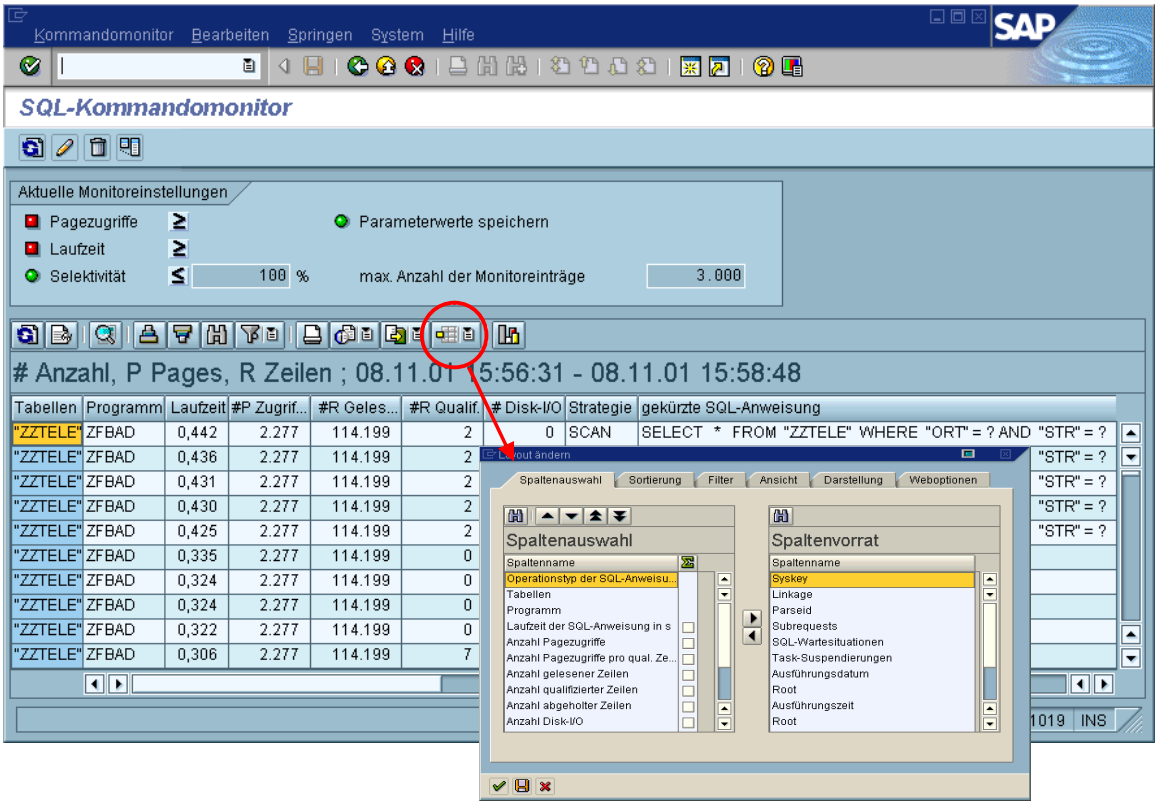
SQL statement runtime: An SQL statement is logged when the runtime exceeds the specified value (in seconds).

Selectivity: An SQL statement is logged in the command monitor tables if the ratio of qualified records to records read falls below the specified percentage.

Save parameter values: Select this field if you want to log the SQL statements with their parameters.

Max. number of monitor entries: This value determines the maximum number of entries that are held in the table SYSMONITOR before the table is overwritten.

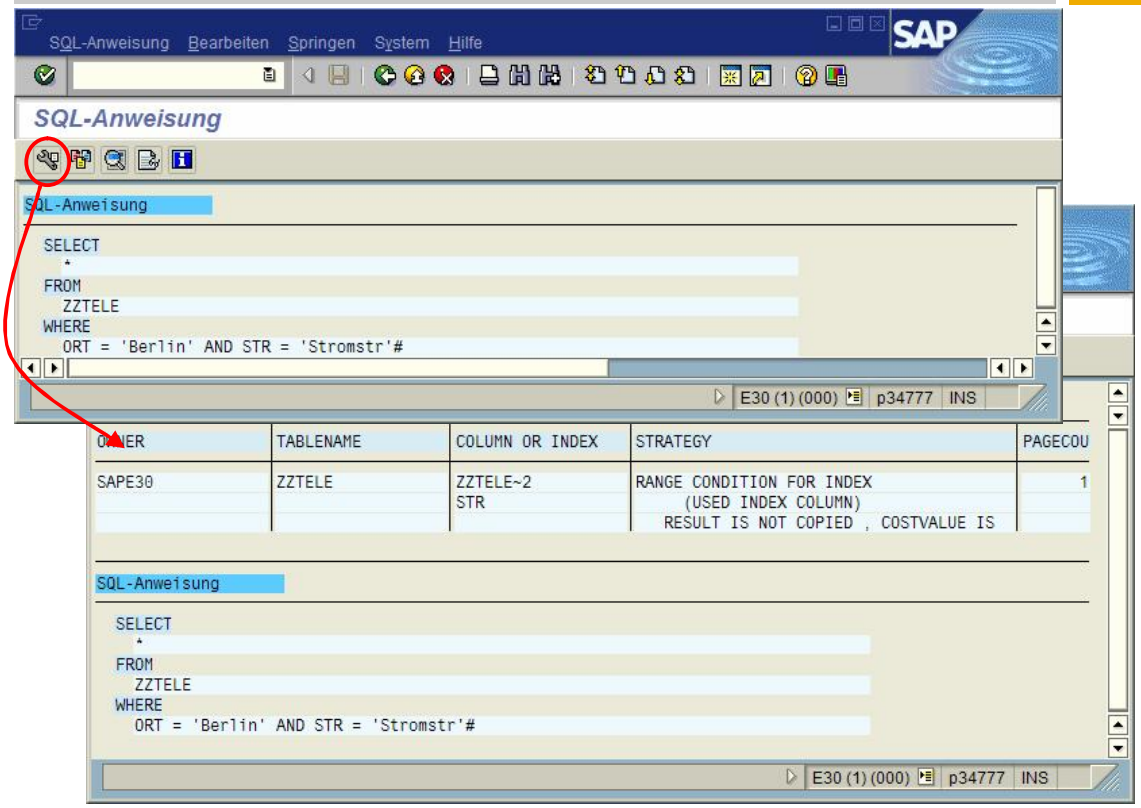
DB50: Problem Analysis – Command Monitor (2)



Description of columns that can be configured via the "Change Layout" button:

Table	Table on which the SQL statement was used
Program	ABAP program that executed the SQL statement The prerequisite is to set the WebAS profile parameter: **dbs/ada/register_appl_info = 1**
Runtime	Runtime of the SQL command in seconds
#P Accesses	Accesses to data pages (in the cache and on the disk)
#R Read	Table rows that were read while processing the statement
#R Qualif.	Table rows that met the selection condition
Selectivity	Ratio #R qualif / #R read in %
#P/R	Number of page accesses per qualif. row
#Fetched (#Abgeholt)	Number of rows fetched
#Disk I/O	I/O accesses to disk (reading and writing incl. converter)
Strategy	The select strategy chosen by the Optimizer (table scan, index, key etc.)
Parseid	Parse ID
SQL wait situations	Number of collisions on SQL locks
Task suspensions	Number of collisions on internal locks
Number fetch requests	Number of fetches while processing the Select statement
Result set	YES, if internal result set was generated (e.g. with sorting), otherwise NO
Date	of execution
Time	of execution
Subrequests	

DB50: Problem Analysis – Command Monitor (3)

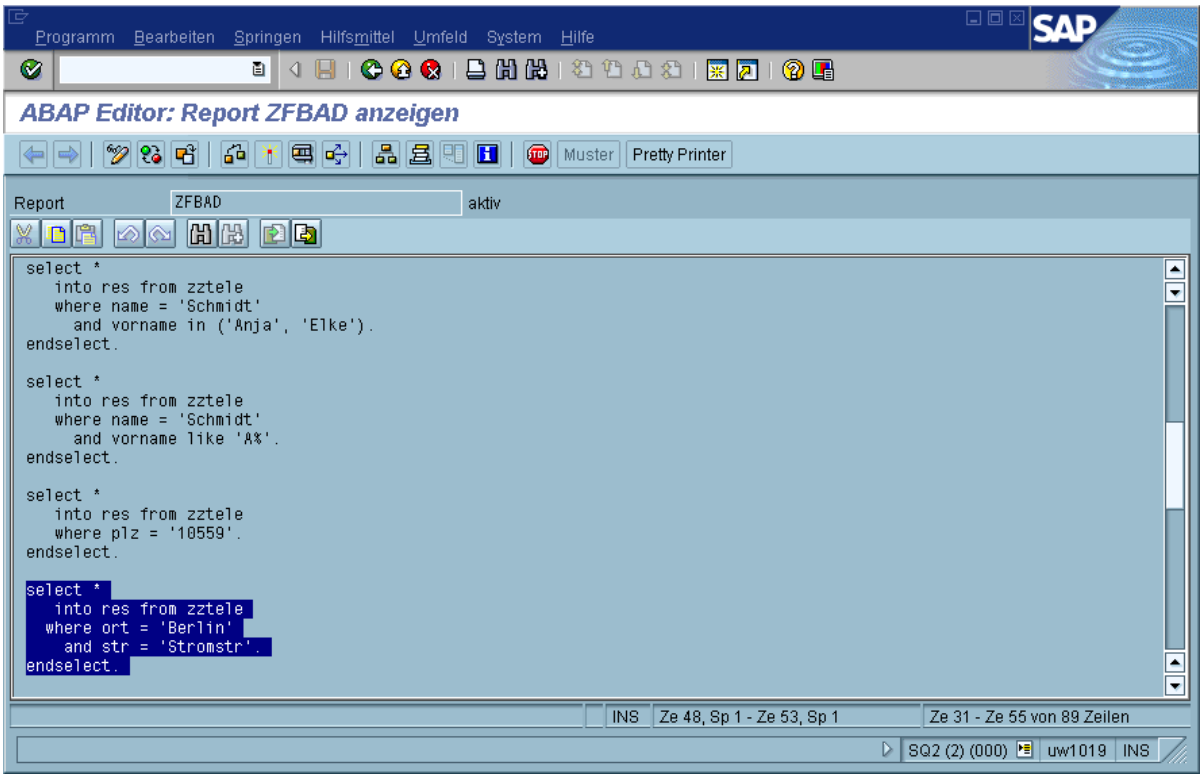


To get the detail view of the command, select and double-click the command you want to examine.

If the *Save parameter value* criterion is active (data collection on), choose *Display Execution Plan for SQL Statement* to check which access strategy the SQL optimizer would choose to process this SQL statement.

If an error in the MaxDB Optimizer is responsible for an unsuitable strategy, development support may require a trace of the call of the Explain statement. To do this, choose *Trace Execution Plan for SQL Statement.* This generates a special Optimizer Vtrace that can be analyzed using transaction DB50 Problem Analysis-> Database Trace.

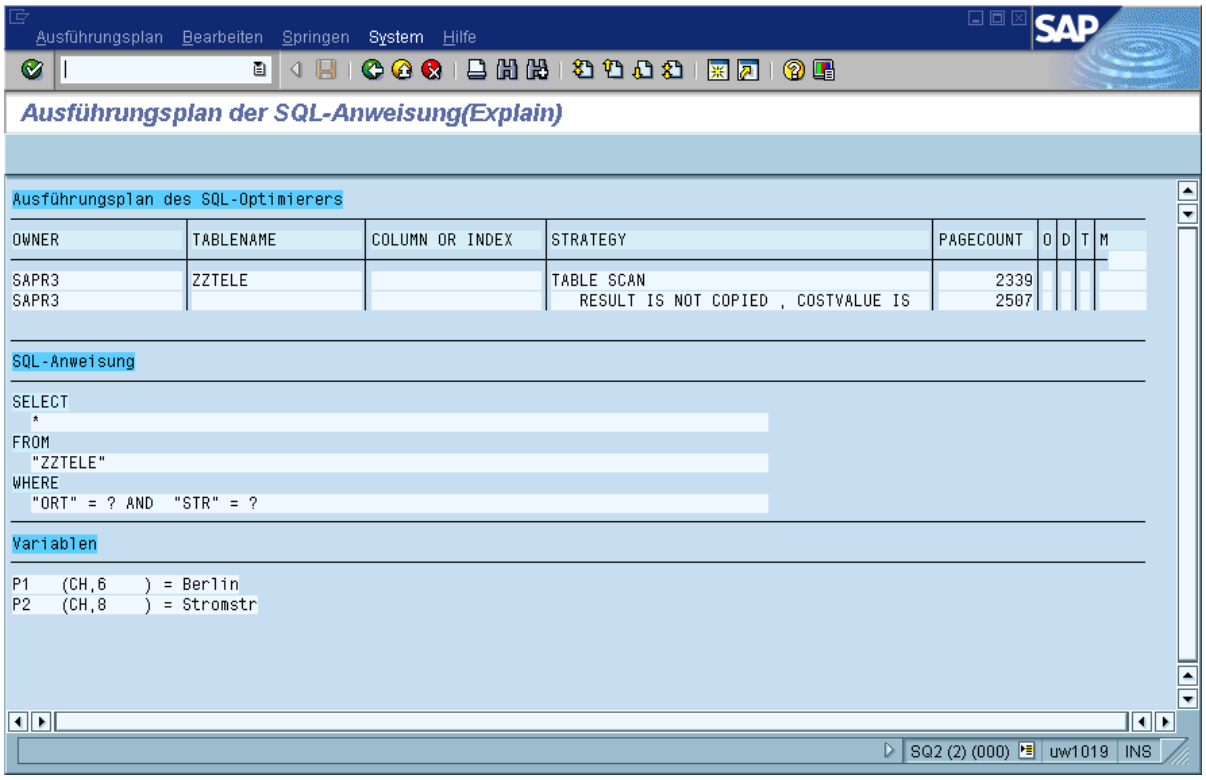
DB50: Problem Analysis – Command Monitor (4)



The *Calling point in the ABAP Program* can be used to determine the BAP program from which the statement was started. For this link to work with systems with WebAS versions below 6.40 dbsl patch 32, set the following parameter in the instance profile of the SAP WebAS: dbs\ada\register_appl_info = 1.

In most cases, the processing of a statement can be accelerated by adjusting the code.

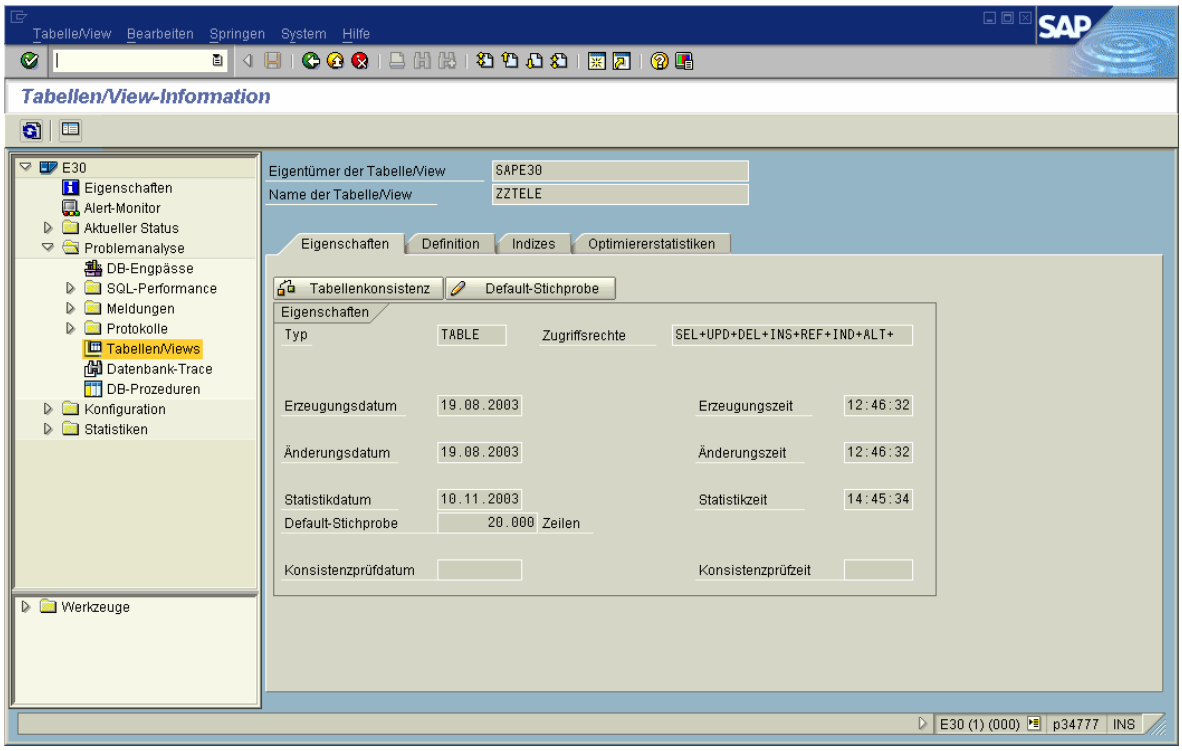
DB50: Problem Analysis – Command Monitor (5)



Using the Explain statement, we see that in our case the statement was executed with a table scan. That is, the whole table was read in order records that have *Berlin* in the *CITY* column and *Stromstr* in the *ST* column.

Here the question arises why an index wasn't used, or rather, why the SELECT was formulated in such a way.

DB50: Problem Analysis – Tables / Views (1)



You can get information on a specified database view or table via DB50 or directly from the command monitor for the current table.

Attributes: Type, access rights, creation and change dates, the date of the last run for determining optimizer statistics for this table, as well as the date of the last Check Table on this table (show table).

Definition: Definition of the table in the database instance (this is not the table definition from the ABAP Dictionary but rather the table definition from the system tables of the database system)

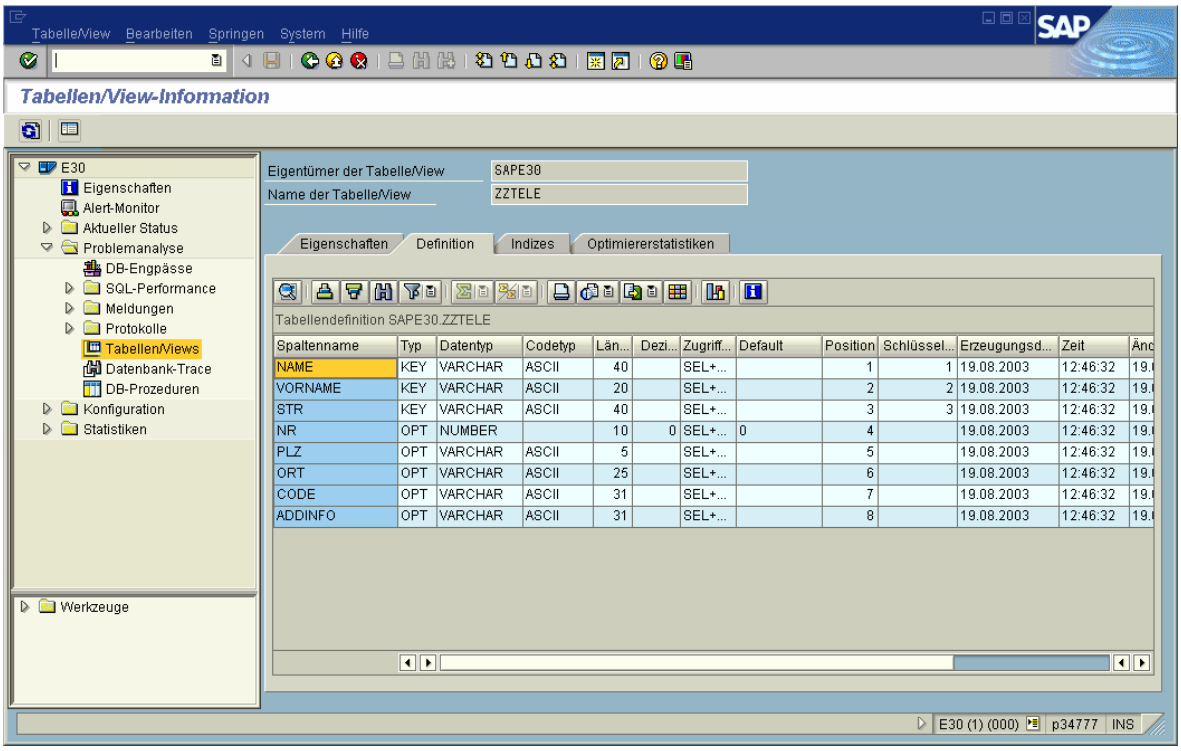
Indices: Indices defined for this table (show index).

Optimizer statistics*:* Last values determined for the optimizer statistics (show optimize stat).

Table consistency: It is possible to start a CHECK TABLE directly from DB50.

Default sample: Using this function, in the system table domain.tables you change the sample value for this table when carrying out the UPDATE STATISTICS command. From then on, all following UPDATE STATISTICS are carried out using the new sample value.

DB50: Problem Analysis – Tables / Views (2)



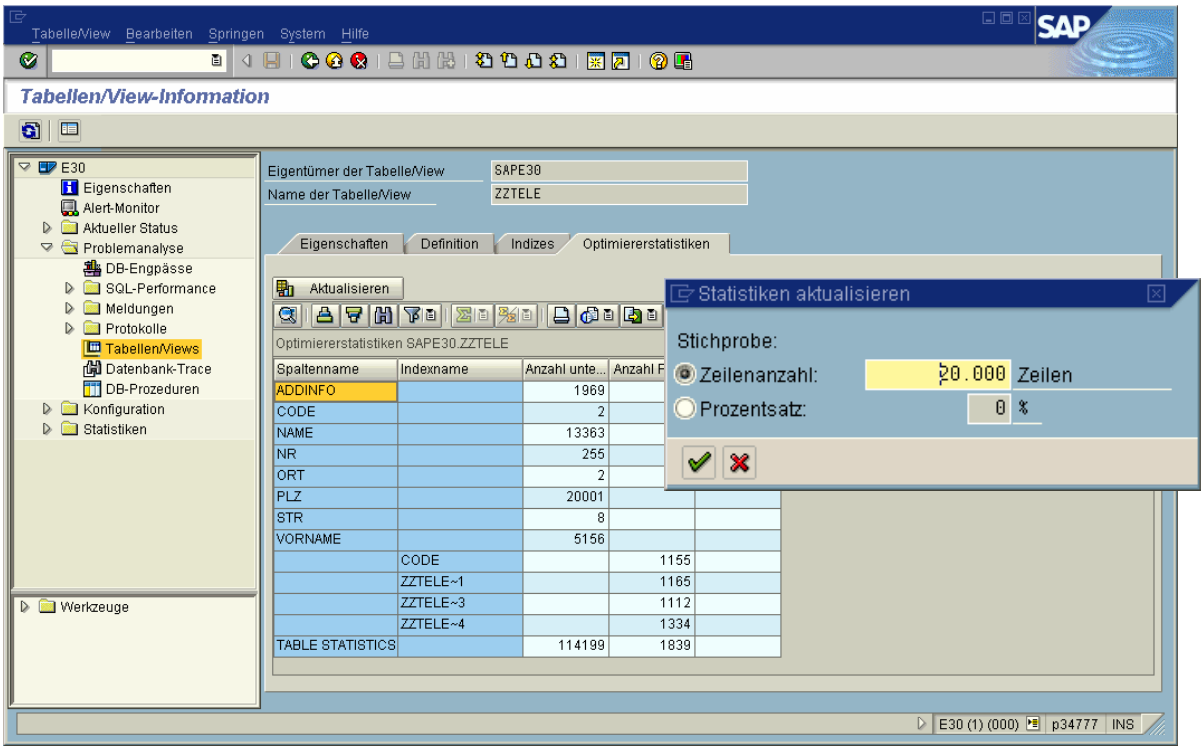
The table definition can provide information as to whether the command could have been processed using the primary key.

The key of table ZZTELE consists of the columns: *Name, First name* and *Street.*

The WHERE condition of the SQL statement consists of the columns *City* and *Street.*

Because neither *Name* nor *First name* are specified in the WHERE condition, the key cannot be used for optimization.

DB50: Problem Analysis – Tables / Views (3)



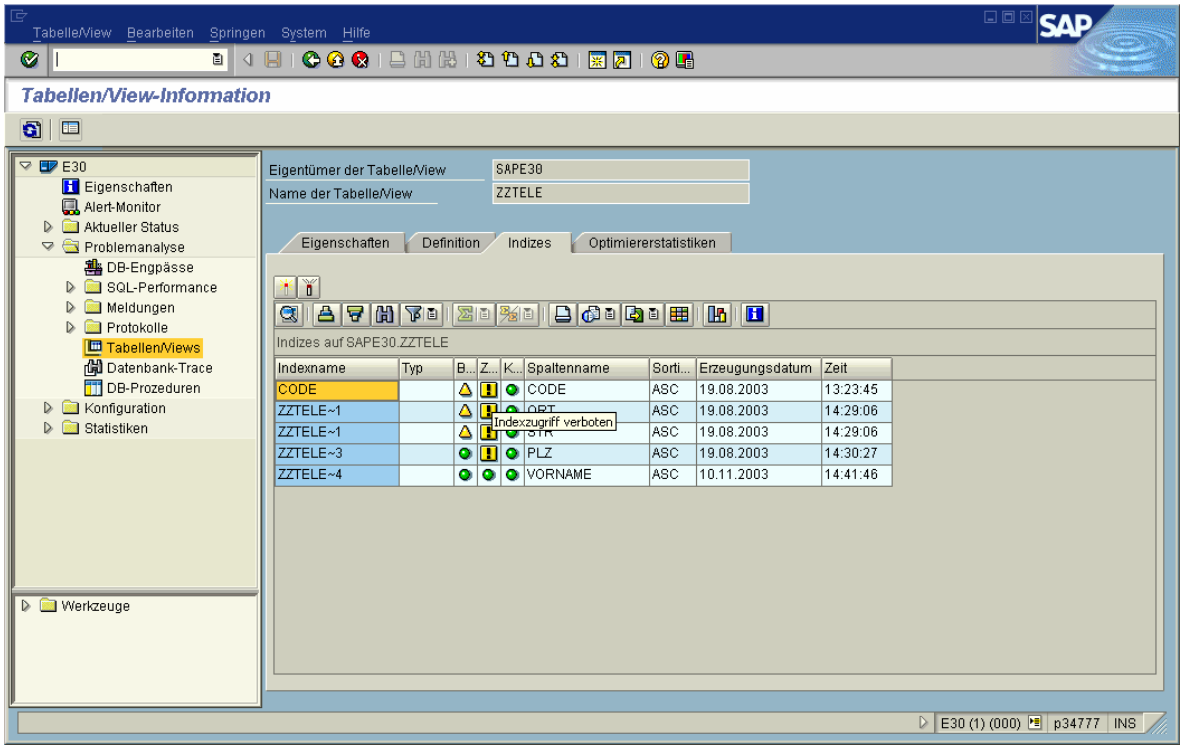
The optimizer statistics provide an overview about the selectivity of the individual columns.

The cost-based optimizer determines the best access strategy with the help of statistical information about the size of the table and values within the table columns.

A cost-benefit plan is created for the various access options.

The optimizer statistics are updated by an UPDATE STATISTICS. You have the option to specify sample values for this UPDATE STATISTICS run. There is no entry of the sample value in the system table *domain.tables.*

DB50: Problem Analysis – Tables / Views (4)



Using the *Indices* function you can check the following:

- whether indices exist for the table
- whether these indices were already used by the optimizer
- whether access to this index is allowed (enabled) or not allowed (disabled)
- whether the index is consistent or perhaps set to BAD
- via which table column the index has been created

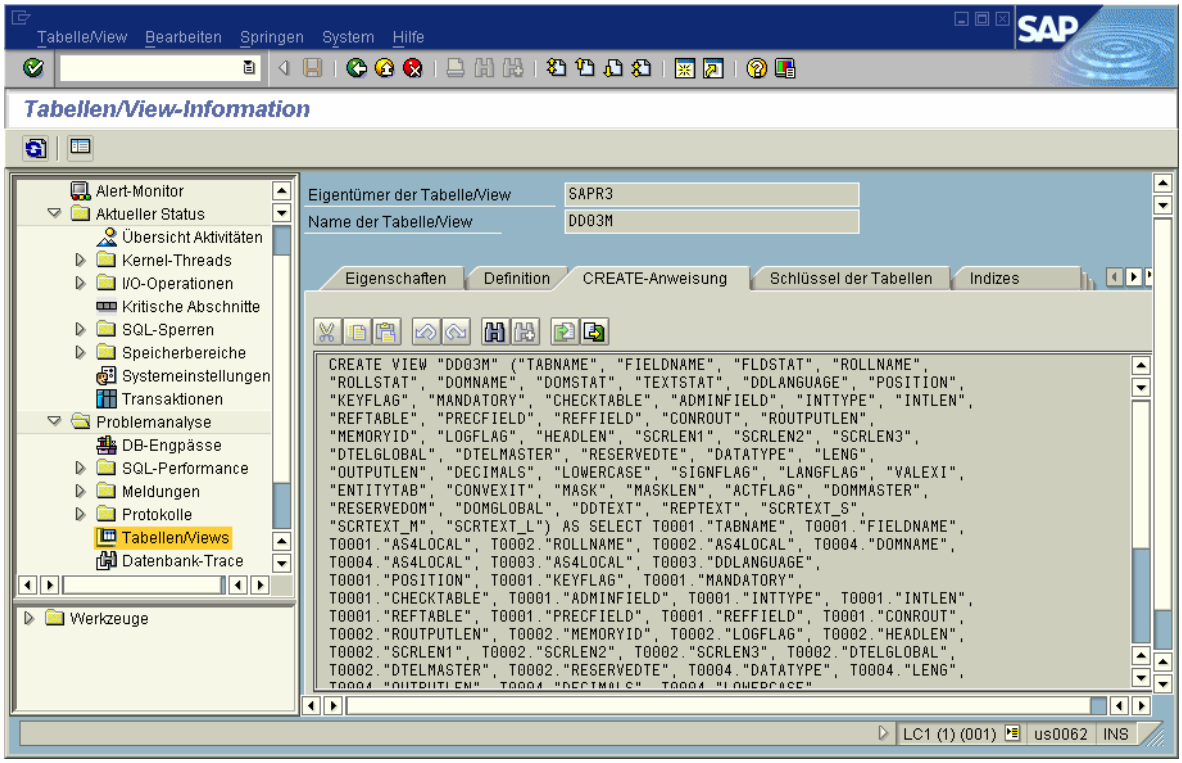
There were 3 indices created for the table ZZTELE, and access to them is forbidden. In other words, the indices were disabled. This could have been necessary in order to test out how a command will be processed if the index did not exist. This feature is offered since this procedure can be performed more quickly than if the index is first deleted and then recreated later. Often this is not possible, especially for large table.

The command logged in the command monitor could therefore only be performed via a table scan because the index that could be used for the optimization is inactive.

The index can be activated directly from this menu. You can do this by selecting the index and choosing *Allow index access.* The column *Access* will be highlighted in green after performing this action.

After restarting the application, the analyzed command may no longer appear in the command monitor.

DB50: Problem Analysis – Tables / Views (5)



As of version SAP WebAS 6.20 with Basis Support Package 39, the detail display is also expanded.

Properties: Type, access rights, creation and change date

Definition: Definition of the tables in the database instance that are involved in the view (this is not the table definition from the ABAP Dictionary but rather the table definition from the system tables of the database system)

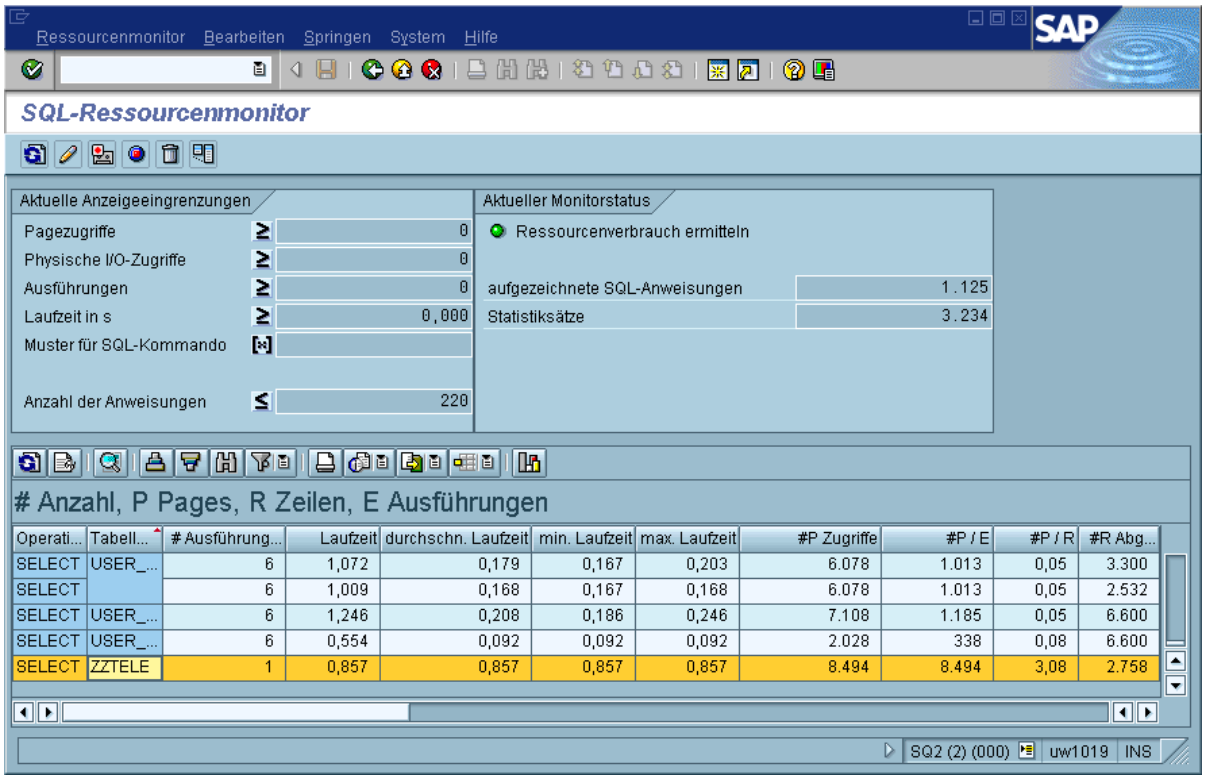
Create Statement: displays the create statement with which the view was created.

Keys of the Tables: all key columns of all tables involved in the view.

Indexes: Indexes defined for this table

Optimizer statistics*:* Last values determined for the optimizer statistics (show optimize stat).

DB50: Problem Analysis – Resource Monitor (1)



By analyzing the resource consumption, you can identify the most costly SQL statements. The resources used by an SQL statement are measured (runtime and I/O accesses, for example).

If an SQL statement is used more than once, the total cost is calculated. This enables you to recognize those SQL statements that have a relatively short runtime, but that generate a considerable database load due to the number of times they are executed.

The resource monitor is therefore a monitoring tool that can be used for load analysis of one workday, for example.

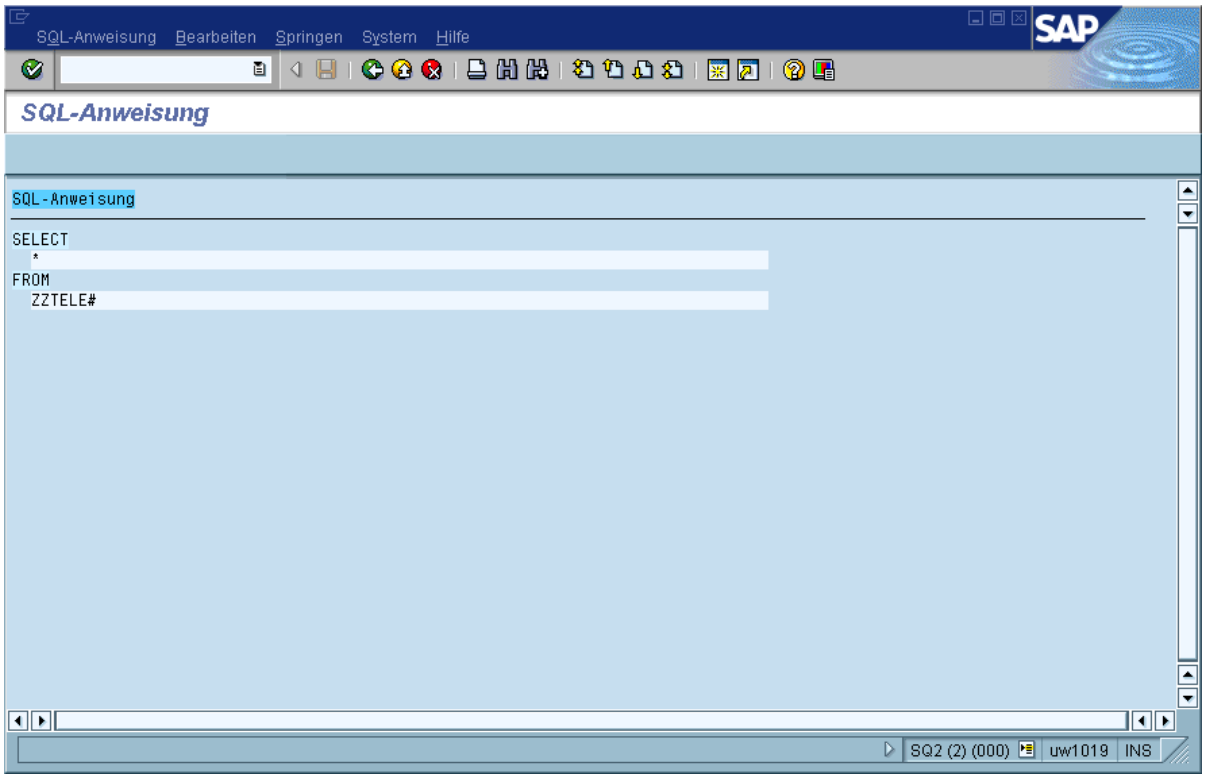
You can restrict the statements to be displayed using additional definitions in the display limits.

With regard to the message in the bottleneck analysis *High Read rate physical* , it is page accesses that interest us here.

What is remarkable is that the last statement was only carried out once but displays over 8400 page accesses.

You can view the statement by double clicking on it.

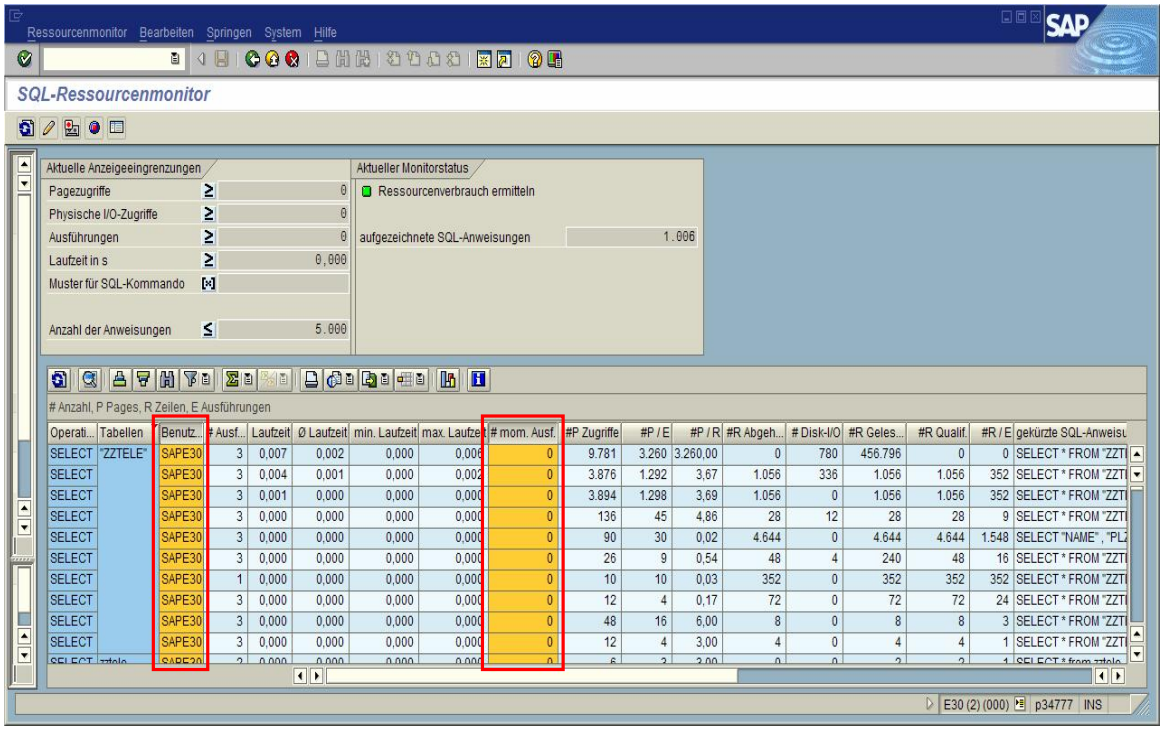
DB50: Problem Analysis – Resource Monitor (2)



Here we have a SELECT for table ZZTELE without a WHERE condition. Therefore all records of the table are read.

In such a case, the application should be examined more closely to see whether all records of the table are truly necessary for processing or whether using a WHERE condition as a limitation, the actual number of records to be processed can be decreased.

Problem Analysis – Resource Monitor with Shared SQL



SharedSQL is activated/deactivated with the parameter SHAREDSQL [YES/NO] or temporarily with "DIAGNOSE SHARE PARSE [OFF/ON]" (only 7.5). The DIAGNOSE command changes the behavior for all sessions that are opened successively. The columns in the resource monitor, however, are displayed corresponding to the parameter SHAREDSQL. After restarting, the parameter value from SHAREDSQL exclusively applies.

With Shared SQL activated, the Resource Monitor offers the additional columns "Users" and "Current executions of SQL statements" as of the following basis Support Packages:
SAP 4.6C = Basis SP48; SAP 4.6D = Basis SP37; SAP 6.10 = Basis SP40; SAP 6.20 = Basis SP40; SAP 6.40 = Basis SP03

Please observe note 767635, which describes from which MaxDB build SharedSQL is recommended. Up to and including 7.5.00 Build 24 , SharedSQL should not be used.

Performance Tables: DIAGNOSE ANALYZE

SYSCMD_ANALYZE

- shows command-ID and SQL command string

SYSDATA_ANALYZE

- entries never deleted or overwritten
- contains command-ID and measured data

every SQL command is recorded

- select * from syscmd_analyze sc, sysdata_analyze sd where sql_statement like '%ZZTELE%' and sc.cmdid = sd.cmdid

Performance Tables

The tables SYSCMD_ANALYZE and SYSDATA_ANALYZE are generated and subsequently filled after DIAGNOSE ANALYZE is activated.

Logging can be activated/deactivated with the following dbmcli command

```
dbmcli –n <SAP DB Hostname> -d <SID> -u control,control -uSQL sap<sid>,sap
        sql_execute  diagnose analyze on | off
```

During parsing, the commands are entered in SYSCMD_ANALYZE and a unique command key is generated. Identical commands are stored only once for all concurrent sessions. Resource usage is not yet determined.

Logging of resource usage can be activated/deactivated with the following dbmcli command

```
dbmcli –n <SAP DB Hostname> -d <SID> -u control,control -uSQL sap<sid>,sap
        sql_execute  diagnose analyze count on | off
```

Normal monitoring is required for this, that is, it is activated if necessary. The values are aggregated per session under the command key in the table SYSDATA_ANALYZE.
Aggregation over several sessions must be done by the application via the command key.

The generated data can be deleted with the following dbmcli command

```
dbmcli –n <SAP DB Hostname> -d <SID> -u control,control -uSQL sap<sid>,sap
        sql_execute diagnose analyze CLEAR COMMAND/DATA/ALL
```

Tools for Updating Optimizer Statistics

Transaction DB13, DB20, DB50

- implemented through a work process with sampling
- SAP WebAS Alert table support

DBMCLI

- sql_updatestat
- sql_updatestat_per_systemtable

Database Manager (DBMGui)

- Instance -> Tuning -> Optimizer Statistics

As of version 7.5, MaxDB only requires statistics data for joins and selects with a restriction of the number of records in the result, such as "WHERE ROWNUM <= n".

For the table itself, Update Statistics only determines the data if the current size specifications are not in the file directory. This does not apply to tables that were created with databases of version < 7.6 and for which the size specifications in the file directory could not yet be determined.

Update Statistics determines statistics data for all columns that are primary keys or index columns. Additionally, it determines the statistics data for all columns beyond the primary key and index if statistics are already available.

If the optimizer discovers tables with unsuitable statistics data, it enters them in the table SYSUPDSTATWANTED. The DBM command sql_updatestat_per_systemtable executes an Update Statistics for all tables listed in SYSUPDSTATWANTED.

The DBM command sql_updatestat executes an Update Statistics for all tables in the database.

Update Statistics imports the data for a table from all data volumes in parallel. This makes it very speedy.

The programs "xpu" and "updcol" are no longer available as of version 7.6..

Sampling Rates with Update Statistics

Sample rates for Update Statistics can be configured as

Rows per table:

- UPDATE STATISTICS ... ESTIMATE SAMPLE <n> ROWS

Percentage per table

- UPDATE STATISTICS ... ESTIMATE SAMPLE <p> PERCENT

Advantage of sampling:

- shorter runtime of update statistic job

Disadvantage of sampling:

- Sample values are only estimated, if they do not resemble the actual data distribution, the optimizer might chose a suboptimal access strategy

Sampling with Update Statistics

Database statistics can be created on the basis of samples. The basis for the statistics can be either a number of rows of your choice or a percentage of the table. The statistics values generated can be queried using SHOW OPTIMIZE STAT <tablename>. While the statistics are not exact, there are generally sufficient for a correct calculation of the SELECT strategy since this depends less on precision than on distinguishing between selective and non-selective columns.

Especially when creating an additional index for an inefficiently processed SQL command, the selectivity of all columns of a table can be determined relatively quickly using ‘UPDATE STATISTICS COLUMN (*) ESTIMATE SAMPLE 20000 ROWS’ . The selectivity of a column is an important criterion when selecting index columns.

The following values have proven adequate sampling quantities for column statistics: 20,000 rows or 10% for tables with more than 1,000,000 data records.

As of version 7.6, the sampling procedure in the standard uses a new algorithm for calculating the statistics data. You can determine the algorithm to be used with the parameter UPDATESTAT_SAMPLE_ALGO. The new algorithm generates more accurate statistics with fewer records read.

Thank you!