MaxDB performance tuning primer

A history of more than three decades has made MaxDB by MySQL a very mature database with a rich tool set for performance monitoring and analysis. Aside from an activity overview screen and the EXPLAIN SQL statement a SQL logger sophisticated monitoring tools are shipped with MaxDB for free. In a small series of articles we'll try to put light on the miracle of performance tuning. Advanced readers will be disappointed by the first introductionary article: skip it and come back at the second article.

It starts with confusion

Performance analysis is a complicated matter. Performance problems always raise a long list of question: was something wrong with the SQL command, was it a high-load situation that caused delays, is the system I/O bound, did I set a database parameter wrong? There's no simple answer to all these questions. The truth is, that databases are large and complicated software products. MaxDB is no exception to this rule. But don't worry, armed with some basic understanding of the implementation of databases you'll be able to get the big picture and learn how individual factors are related. This understanding is a key for a successful database performance tuning. Let's start with a confusing excursus into database implementations in order to proof the said.

In 1973 Michael E. Senko published an article on “Data Structures and Data Accessing in Data Base Systems – Past, Present, Future” in the IBM Systems Journal which sketched the cornerstones of a five layer based database system. The article described the system architecture used for the IBM System R prototype. It has gained some popularity and it has influenced database implementations until today. Although the five layers found by Senko do no map directly to the MaxDB layers, the article is still an excellent resource to identify starting points in performance analysis.

Senko has identified the following layers.

1. Logical Data Schema
   Components: SQL parsing, Access path selection, Access control, Constraints
2. Data Access Management
   Components: Data Dictionary, Transaction handling, Sorting
3. Internal Storage Management
   Components: Record Manager, Lock management, Log/Backup/Recovery
4. Buffer management
   Components: Page Caching
5. Operating system
   Components: external (permanent) storage management

You can think of the first layer as the user interface of the database and the user. The language used for communication between the user and the database is SQL. SQL is the users connection to the data inside the database. The first layer translates the SQL commands of the user into an internal format that can be used inside the database for request processing. But the first layer does more than the SQL parsing. Before any SQL commands can be executed access permissions need to be checked. This check is necessary for all SQL commands, be it a SELECT, a data manipulation query (INSERT, UPDATE, DELETE, COMMIT, ROLLBACK, ...) a data definition query (CREATE TABLE, CREATE SEQUENCE, DROP...) or a data control command (GRANT, REVOKE).

Depending on the type of the SQL command more tasks are handled inside this layer. All statements require a translation the set oriented SQL language used by the user into the record oriented interface used inside the database. For SELECT statements an access path needs to be calculated, an algorithm to find the requested set of data by reading as few rows as possible needs to be calculated. DELETE and UPDATE require the calculation of an access path as well. Before a record can be removed of changed it needs to be found. The requested changes must not be applied Once the records are identified constraints must be checked before the data gets actually changed.

Let's summarize: SQL is the language of communication and SQL needs to be compiled efficiently into a record oriented interface used inside a database. Changes to SQL commands and additional search structures, can lead to significant better compilations and faster access paths.

Every compilation takes time. The more sophisticated rules for optimization are used during the compilation, the longer it takes. The time required for parsing and compilation can get significant. To save the time required for repeated compilation during multiple executions of the same SQL query, MaxDB offers a cache to store parse results. If you see a bottleneck in parse times, you can exchange faster execution and saved parse times against higher memory requirement for the caching.

From this first layer we have learned about two possible starting points: SQL optimization and database parameter tuning. We'll have a broader look at the remaining layers in Senko's model, but you should nevertheless get the point that knowing how a database works internally is a key in understanding performance problems.

The second layer suggested by Senko is the Data Access Management layer. Access does not refer to permissions but to the how the access inside the system is done. The data of the user is modeled in a logical schema that is not related the internal representation of the data inside the database management system. The Data Dictionary (Database Catalog) contains all informations on how the data is structured from the users point of view (Views, Tables, Columns) and how this maps into the internal structure. But not only meta-data on the data format is stored in the Database Catalog. The catalog contains also stored procedures and other SQL elements defined by the user. Once the internal rows have been identified, the transaction management inside the second layer takes care of concurrent accesses to them. It takes care of the redo and undo logfiles, internal locks and it contains the deadlock detection.

Database Catalog accesses are speed up by MaxDB with help of a database catalog cache. If an object cannot be found in the cache a slow disk access is required to load the necessary informations. Remember that this disk access does not fetch any data that has been requested by the user but it's reading only metadata. This is of course not very desirable and you should check during your performance investigations that the catalog cache hit rate is above 85%. How comes you can ignore such a low figure? Problems that come from the sketched theoretical model of Senko do not necessarily apply for MaxDB. MaxDB has been under steady development for many years and many clever, partly unique solutions have been added to the product to face possible bottlenecks. If MaxDB runs out of space in the database catalog cache it starts using space of the data cache. This is of course not the best solution but it's far better than doing a disk access! By the way, this is similar to what the InnoDB storage engine of MySQL does if it runs out of innodb_additional_mem_pool_size and it's starts to allocate memory of the operating system with the difference that MaxDB remains to use only previously assigned memory areas.

Part of the transaction handling is the handling of redo and undo logs. Performance bottlenecks can occur if log write queue overflows and asynchronous I/O cannot be used any more. Locks need to be set and released during transactions to ensure the ACID (atomicy, consistency, isolation, durability) principles. Wait situations can lower the parallelism and the transactional throughput. There's a whole bunch of database parameters more or less directly related to the transaction management that influence the system performance.

In the middle of the architecture sketched by Senko is the third layer, the internal storage management. This layer has finally left the users view on the data and deals with B-trees, database pages and their access. A typical operation inside this layer is to fetch and store a database page for processing in other layers. As it's the central component to deal with one of the smallest logical units databases use internally (a page – memory block of typically 8-16kb) is has to lock pages during modifications to protect a modification that's done by one task to be overwritten at the same time by another task.

A central component in the MaxDB architecture is the converter. The converter contains a mapping table between logical (page numbers) addresses and physical (block/disk) addresses. For the sake of Senko's model your can think of it as a cache and the converter is actually stored in the I/O buffer cache of MaxDB. The converter gets loaded during database startup so that you automatically get a hit rate of 100%. Accesses to the converter need to be synchronized. To avoid collisions so called regions can be defined that break the converter into smaller units which lowers the likelihood of concurrent accesses and locks on the same parts of the cache.

The last layer above the operating system is the buffer management. The buffer management is responsible for an effective loading and storing of database pages and their caching. It's also the point where logical pages are broken up into blocks and smaller units that can be read from the permanent storage media.

This layer icontains the most important cache of the MaxDB database, the data cache. If ever possible, physical read and write operations are avoided in favour of cache accesses. Disk accesses are about 1000 times slower than main memory accesses. The hit rate of the data cache is well worth the very first look during a performance analysis. If anyhow possible it should be around 99% on a long running system.

Finally the operating system offers the necessary interface to the buffer management of the database to read and store data from the storage hardware of the computer.

For cases when MaxDB cannot avoid slow disk accesses like during save points it tries to group pages in order to execute as few I/O operations as possible. Thought this technique is used to minimize I/O times, it shows another trick used by highly optimized software. Operating system calls and switches between the application and the operating system can be considered as expensive. Here's an example.

Databases depend on the speed and features of the operating system like every other software. For example, the speed of the thread library used by the operating system heavily influences the speed of the MySQL server. Each connection, each user task is handled in it's own thread. MaxDB can be configured to run in a similar way and use a thread for each user task or, which is the suggested way, use it's own internal mechanism for user tasks. The latter should give you a performance benefit of some 10%. Factors that need to be considered for MaxDB are spinlocks and asynchronous I/O to speed things up.

On the hardware side the CPU usage and the I/O performance needs to be monitored.

The lessons learned

What's the lesson from this lengthy introduction? Performance tuning is a very complex work. Many, many factors need to be considered and you need a basic understanding of how databases work and how your database in particular is implemented. It's not sufficient to know the name of the performance monitoring and tuning tools that come with MaxDB. Try to get a basic understanding of all components involved in a query: the hardware, the operating system, the database, the client application and it's SQL commands. Though it's a complex task, it's a challenging task and it's well worth the trouble.

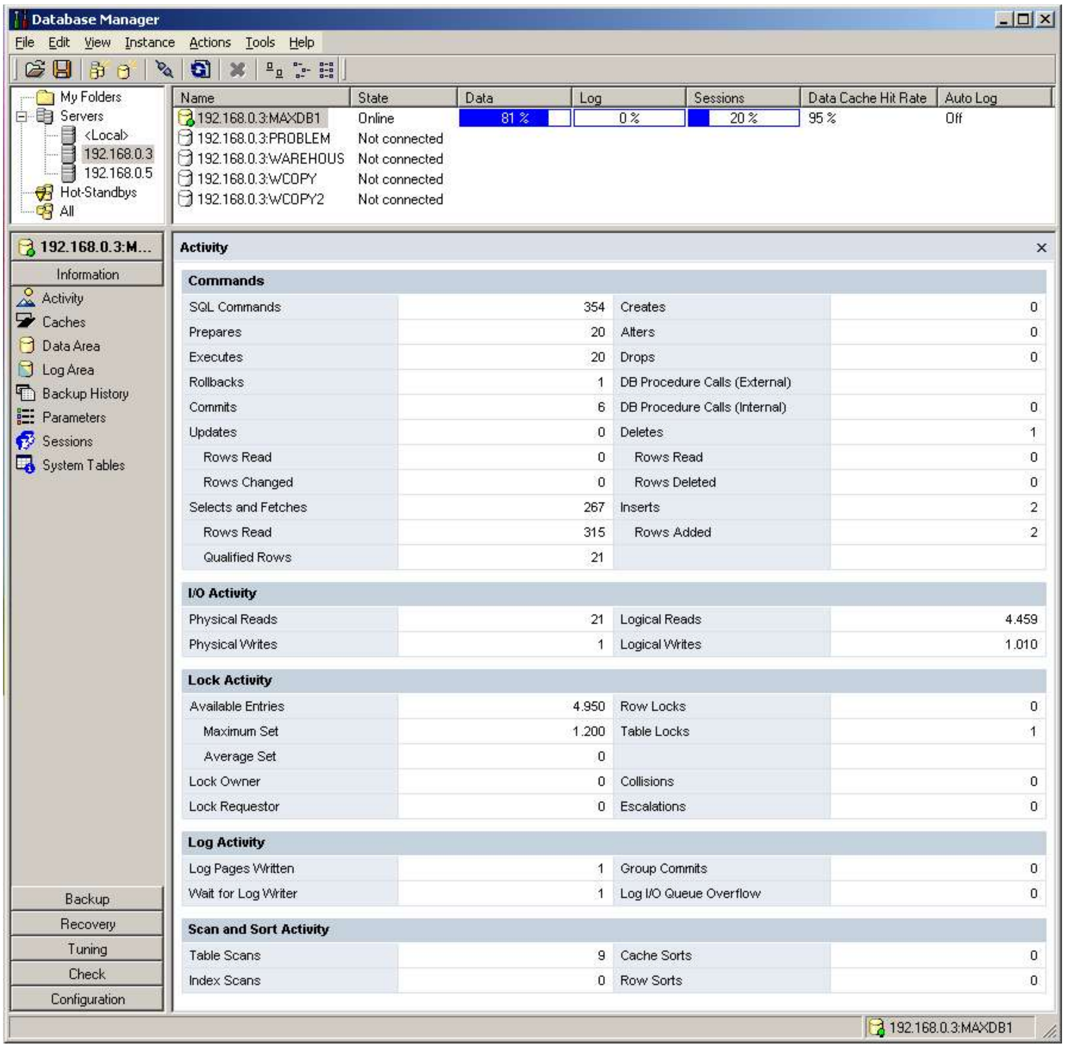
The MaxDB tools

MaxDB comes with five tools for performance analysis. The tools can be split up into two groups for ad-hoc analysis and long term investigations. For ad-hoc analysis the SQL command EXPLAIN and the command line tool x_cons can be used. EXPLAIN [VIEW] SELECT is used to display the execution plan of a SELECT command and check the estimated “costs” for executing the command. The command line tool x_cons takes a snapshot of the database activity. Snapshots can be made of tasks, sessions, parameters and much more.

Long term observations can be made with the Database Analyzer, the Diagnose Monitor and the Diagnose Analyze tools. The Database Analyzer calls the x_cons utility in regular intervals and logs the results. Rules can be defined to assign warning levels to individual figures collected if required. The Diagnose Monitor is comparable to the MySQL Server slow query log. By help of the utility all queries can be logged that exceed certain limits, e.g. a maximum runtime or a maximum number of virtual read operations. With help of the last tool, the Diagnose Analyzer tool, all SQL queries and their performance characteristics can be recorded no matter if they exceed a limit of whatever kind or not.

What's up with my database?

The summary screens of the Database Manager GUI cannot be considered as a performance analysis tool, but they give you a brief summary of what's up with your database. This summary can contain a bunch of hints what you should look for first using the above mentioned tuning utilities. Two screens in the Information section are of interest for you: Activity and Caches.



Name	State	Data	Log	Sessions	Data Cache Hit Rate	Auto Log
192.168.0.3:MAXDB1	Online	81 %	0 %	20 %	95 %	Off
192.168.0.3:PROBLEM	Not connected					
192.168.0.3:WAREHOUS	Not connected					
192.168.0.3:WCOPY	Not connected					
192.168.0.3:WCOPY2	Not connected					

Activity

Commands			
SQL Commands	354	Creates	0
Prepares	20	Alters	0
Executes	20	Drops	0
Rollbacks	1	DB Procedure Calls (External)	
Commits	6	DB Procedure Calls (Internal)	0
Updates	0	Deletes	1
Rows Read	0	Rows Read	0
Rows Changed	0	Rows Deleted	0
Selects and Fetches	267	Inserts	2
Rows Read	315	Rows Added	2
Qualified Rows	21		

I/O Activity			
Physical Reads	21	Logical Reads	4.459
Physical Writes	1	Logical Writes	1.010

Lock Activity			
Available Entries	4.950	Row Locks	0
Maximum Set	1.200	Table Locks	1
Average Set	0		
Lock Owner	0	Collisions	0
Lock Requestor	0	Escalations	0

Log Activity			
Log Pages Written	1	Group Commits	0
Wait for Log Writer	1	Log I/O Queue Overflow	0

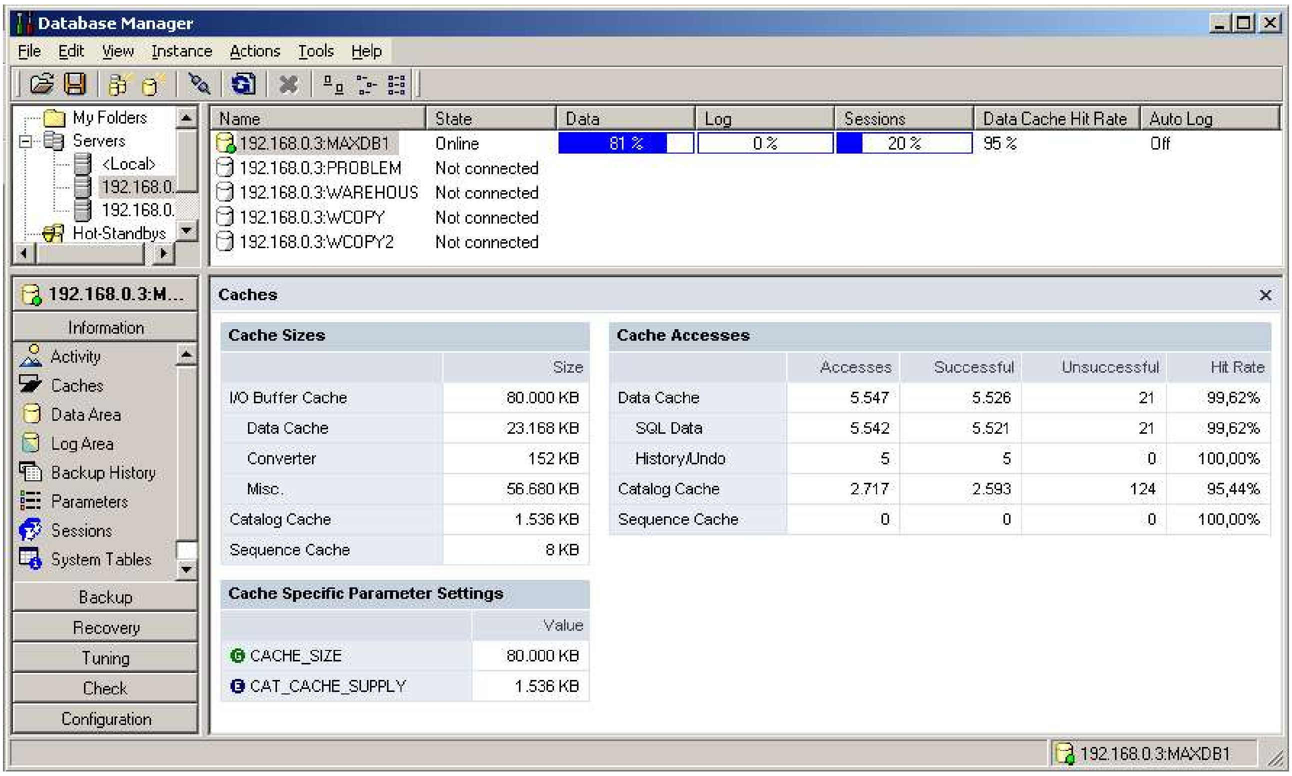
Scan and Sort Activity			
Table Scans	9	Cache Sorts	0
Index Scans	0	Row Sorts	0

The Activity screen is divided into five sections: Commands, I/O Activity, Lock Activity, Log Activity, Scan and Sort Activity. The sections show aggregated figures since the last start of the database instance. In the example pictures most values are 0, because no activity was made against the database after starting it.

The Command section tells how many SQL commands of a certain type have been executed. For example, the number of Prepares tells you how many prepared statements have been sent by the database clients. If the number of Prepares is close to the number of Executes, then you don't profit from the advantage of prepared statements. Prepared statements can save SQL compiling time.

A high number of Collisions in the Lock Activity screen shows you that many concurrent accesses to the same SQL objects have been made. This can lead to wait situations. Check the number of Available Entries, the Maximum Set and the Average Set. If required, increase the number of Available Entries. The number of Escalations shows in how many cases row locks got converted into table locks. A high number of Table Locks decreases the concurrency of your system. Table Locks mean that you can't execute two write queries at once on a table. This can be a negligible if you know from the Commands section that your database is a read-mostly database, but it can be also a critical observation that needs your attention.

One of the most interesting figures in the the Log Activity section is the figure Log I/O Que Overflow. If the que flows over, transactions have to wait for slow physical I/O operations. The last section, Scan and Sort Activity shows how many Index- and how many Table Scans have been done. A high value of table scans is an indicator for many queries that do not use indexes at all or for indexes with a very low selectivity. The two sort statistics document how often sort operations could be performed in the main memory cache and how often the database had to materialize temporary result tables on disk and perform a much slower disk sort. But be careful that you do not get mislead by the figures. Internal sorting operations that are required during the processing of aggregate functions and joins are also count.



Name	State	Data	Log	Sessions	Data Cache Hit Rate	Auto Log
192.168.0.3:MAXDB1	Online	81 %	0 %	20 %	95 %	Off
192.168.0.3:PROBLEM	Not connected					
192.168.0.3:WAREHOUS	Not connected					
192.168.0.3:WCOPY	Not connected					
192.168.0.3:WCOPY2	Not connected					

Caches

Cache Sizes

	Size
I/O Buffer Cache	80.000 KB
Data Cache	23.168 KB
Converter	152 KB
Misc.	56.680 KB
Catalog Cache	1.536 KB
Sequence Cache	8 KB

Cache Accesses

	Accesses	Successful	Unsuccessful	Hit Rate
Data Cache	5.547	5.526	21	99,62%
SQL Data	5.542	5.521	21	99,62%
History/Undo	5	5	0	100,00%
Catalog Cache	2.717	2.593	124	95,44%
Sequence Cache	0	0	0	100,00%

Cache Specific Parameter Settings

	Value
CACHE_SIZE	80.000 KB
CAT_CACHE_SUPPLY	1.536 KB

The Cache overview section is fairly simple to understand. MaxDB does not bother the Database Administrator with many, fine-grained settings for Caches, Logfiles and Datavolumes. MaxDB makes the assumption that you always have enough main memory available to archive good overall Cache Hit Rates and you don't need to care on how to split up a short amount of main memory for individual tasks. This is similar to the assumption made about Logfiles and Datavolumes. MaxDB always stripes data evenly over all available Datavolumes. You cannot bind a table to a certain datavolume. The philosophy of MaxDB is that if you need maximum speed you should go for a Storage Area Network Solution which offers more sophisticated algorithms than any database to archive a maximum I/O performance. However, this philosophy of MaxDB makes your work easier. All you need to do is to check the Hit Rates. You can relax if the database cache hit rate is around 99% and the Catalog Cache is close to 85%. If it's far below, your system will be very likely I/O bound.

So much about the basics, it's time for a first hands-on on the five most important performance tuning tools.

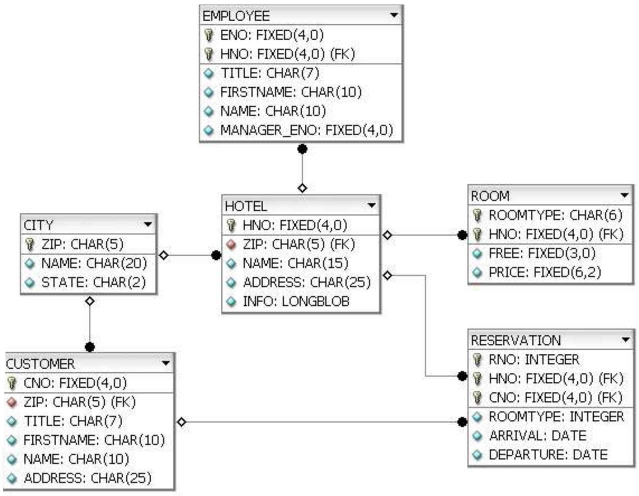
SQL Optimization with EXPLAIN

If any of your database programmers is unable to read and understand the EXPLAIN output of the database system you're using, send him to class and teach him how to do it. Remember that SQL is language you speak to your database. The better the database understands you and the better the database is prepared for your queries, the faster you'll get your data out of the database. EXPLAIN tells you how your commands have been executed by the database. The execution plan EXPLAINS shows unveils how well the database has understood your SQL. Like in other business areas it's necessary that you check how efficient your database programmers and the particular database product communicate with each other. In some rare cases it might be required that train your programmers not only in SQL-92 (Oxford english) but also in the SQL dialect of the database (Irish) – however, let's get back to EXPLAIN the tool to unveil slow execution plans and misunderstandings.

MaxDB knows of two variants of EXPLAIN: EXPLAIN and EXPLAIN VIEW. EXPLAIN can be used to display execution plan details on SELECT statements and other “query_statements”. In the MaxDB syntax query_statement are one of simple SELECT statements, named SELECTs, DECLARE CURSOR and recursive DECLARE CURSOR statements. This is a little bit different to the MySQL Server where EXPLAIN has to be followed by a SELECT statement or by an object name if you want to use it to display CREATE informations. EXPLAIN cannot be used to print the execution plan of DML and DDL queries like it's possible with the Oracle statement EXPLAIN PLAN. We'll show you a trick in the next performance article how you can work around this restriction.

The only difference EXPLAIN and EXPLAIN VIEW is that EXPLAIN VIEW gives one additional information. EXPLAIN VIEW shows the names of base tables hidden inside a view definition and not only the name of the view as EXPLAIN does.

For practical example we'll use the complete version of the HOTELDB database schema. The HOTELDB is the tutorial database schema used in the SQL tutorial (see Getting Started on the Documentation overview page http://dev.mysql.com/doc/maxdb/index.html) and during the MySQL MaxDB training courses. The ER model is rather simple. We won't explain it because we're primarily interested in EXPLAIN command and not in the ER model.



For the first steps with EXPLAIN a fairly simple query is used: EXPLAIN SELECT name FROM city WHERE name > 'H' . When executed MaxDB prints a result table with five columns. For each table contained in the SELECT statement one row is added to the result table. The column OWNER shows the owner of the table that's shown in the column TABLENAME. COLUMN_OR_INDEX shows the name of a key- or index column if one was used to scan a table. The next column STRATEGY tells you which algorithm, which strategy was used by the optimizer. The last column PAGECOUNT show how many pages a table occupies. Note that this figure does not tell you how many pages need to be read during execution of the SELECT statement!

OWNER	TABLENAME	COLUMN_OR_INDEX	STRATEGY	PAGECOUNT
DBA	CITY		TABLE_SCAN	1
		RESULT IS NOT COPIED, COSTVALUE IS		1

The last row tells you if a temporary table, a temporary result set needs to be build and gives and estimation of the number of virtual I/O operations which would be required to execute the the SELECT statement. The COSTVALUE cannot be compared between major database versions, because the Optimizer uses different algorithms to calculate it in different versions. Keep this in mind before you turn to the support channels complaining about bad performance after a version upgrade ;-). Though we speak about I/O operations in this text we do not mean function calls but pages.

The COSTVALUE is no more but a guess on how many pages would be read and written if the SELECT statement would have been executed. EXPLAIN never really executes the SELECT statement but only calculates the execution plan it would use. This is also the reason why the number of required page (I/O) operations does not give any hints how many physical I/O operations are required and how many pages can be fetched from the caches.

Nevertheless the COSTVALUE is the most important figure followed by the STRATEGY column. In the above EXAMPLE a TABLE_SCAN strategy gets used. A table scan tends to be a sub-optimal way to search for data. Each row of the city table is checked against the condition name > 'H'. But in this case it's the fastest possible solution and it's not even a bad choice. No other strategy can be used, because no other search structures, no keys no indexes are available for the name column of the city table. Even if an index would exist its likely that MaxDB would not use it. For small tables or indexes with bad selectivity it's more complicated to do the two step lookup required for an index (first step: find the primary keys by reading the index, second step: find the data using the primary keys).

The next EXPLAIN example shows two rows for two tables contained in the JOIN of the hotel and city table that's used to display all hotels in the zip code range between 10000 and 10100. The SQL query is:

```
EXPLAIN
  SELECT
    c.name AS cityname,
    c.zip AS zip,
    h.name AS hotelname
  FROM
    city AS c,
    HOTEL as h
  WHERE
    (c.zip >= '10000' AND c.zip <= '10100') AND
    (c.zip = h.zip)
```

OWNER	TABLENAME	COLUMN_OR_INDEX	STRATEGY	PAGECOUNT
	H		TABLE_SCAN	1
	C	ZIP	JOIN VIA KEY COLUMN	1
NO TEMPORARY RESULTS CREATED				
RESULT IS COPIED, COSTVALUE IS				2

The first rows shows the first table that was investigated by MaxDB. The order of the tables is equal to the order of execution. MaxDB investigates the hotel table first and does a full table scan on it. Then it joins the city table via the primary key column zip. In total the optimizer makes the guess that 2 I/O operations are required to calculate the result and prints this in the COSTVALUE line.

We'll stop the discussion of EXPLAIN. EXPLAIN and the MaxDB optimizer are documented in great depth in the manual. Check the section Background Knowledge => SQL Optimizer for more details and wait for the other articles in our small performance tuning article series.

Snapshots with x_cons

All statistical informations about the database activity are collected from a rich set of system tables and the x_cons utility. x_cons is a mighty command line tool that can only be run on the database host.. If you don't have access to a shell on the database host, you can use the dbmcli utility to call it remotely: dbmcli -d ... -u ... -n <host> db_cons <command>. We recommend x_cons only for use by support workers, developers and very experienced users.

The help output shows some 25 commands. This is of course far to much to be discussed in this introduction.

```
nixnutzlinux:/opt/sdb/programs/bin # ./x_cons MAXDB1
Command: help
usage: x_cons [<serverdb>] <Command> [<Interval> [<Repeat>]]

 SERVERDB                    ::  <serverdb>

 Commands:
  time measurement           ::  TIME   <ENABLE | DISABLE>
  cancel the command of task ::  CANCEL <taskindex>
  release cached system pages::  PAGECACHE_RELEASE
  diagnose system page cache ::  DIAGNOSE_PAGECACHE
  kill the session of task   ::  KILL <taskindex>
  flush trace file           ::  TRACE_FLUSH
  show statistics/states     ::  SHOW IO
                                 SHOW AIO       (backup only)
                                 SHOW DEV_IO    (volume only)
                                 SHOW STORAGE
                                 SHOW TASKS     [ DW | SV | US ]
     alternative for TASKS   ::  SHOW PROCESSES [ DW | SV | US ]
                                 SHOW ACTIVE    [ DW | SV | US ]
                                 SHOW RUNNABLE  [ DW | SV | US ]
  show move info (load bal.) ::  SHOW MOVEINFO
  show task counts           ::  SHOW T_CNT     [ DW | SV | US | T<taskindex>]
  show task queues           ::  SHOW T_QUEUE
  show task regions          ::  SHOW T_REG
  show task statistics       ::  SHOW T_STAT
  show tasks move info       ::  SHOW T_MOVE
                                 SHOW VERSIONS
                                 SHOW REGIONS
                                 SHOW STATE
                                 SHOW RTE
                                 SHOW QUEUES
  suspend reasons            ::  SHOW SUSPENDS
  UKT sleep statistic        ::  SHOW SLEEP
  Thread stacks              ::  SHOW TSTACK
```

```
                                SHOW ALL
Command: quit
nixnutzlinux:/opt/sdb/programs/bin #
```

In order to demonstrate how detailed the informations are that can be queried with x_cons, we'll have a brief look at the SHOW ACTIVE and T_C commands. SHOW ACTIVE prints a list of all task currently running. The output shows one user task connected. The task has the internal ID T45 and it's currently running. The task has been created by an application with the process ID 1480*. The asterix means, that the application client is running on another host.

```
nixnutzlinux:/opt/sdb/programs/bin # ./x_cons MAXDB1 SHOW ACTIVE

SERVERDB: MAXDB1

ID   UKT UNIX   TASK       APPL Current         Timeout Region     Wait
          tid   type        pid state          priority cnt try    item
T19    5     -1 EventTs    8346 Running               0 0               664(r)
T45    7     -1 User      1480* Running               0 40              3762(r)
```

By help of the T_C <task> command one can get detailed measurements of an individual task. If you compare multiple output of T_C for the same task you'll see that the informations displayed vary from time to time. Some informations make only sense for developers, some are of general interest. For example the rcv_rpl_long value shows that 2 queries have been executed with a runtime of more one second but the average kernel execution time of all queries handled by the task (avg_rcv_rpl_t) is only 0.0994 seconds.

```
nixnutzlinux:/opt/sdb/programs/bin # ./x_cons MAXDB1 SHOW T_C T45

--------------------  T45   User      ( pid =   1480 ) -----------------------
 remote_node   : CPQ31170544531                        remote_pid    : 1480
 dispatcher_cnt: 23200                                 command_cnt   : 2263
 exclusive_cnt : 6281463                               self_susp_cnt : 20441
 Resume count 0  total 273        History [ T2 T2 T2 ]
 self_read_io  : 235                                   avg_self_rd_tm: 0.0000
 self_write_io : 6          self_write_pg : 6          avg_self_wr_tm: 0.0090
 dev_read_io   : 116        rel_dev_rd_tm : 0.0007     abs_dev_rd_tm : 0.0007
 dev_read_pg   : 116                                   pages_per_io  : 1.0
 dev_write_io  : 39         rel_dev_wr_tm : 0.0022     abs_dev_wr_tm : 0.0022
 dev_write_pg  : 39                                    pages_per_io  : 1.0
 rcv_rpl_count : 659        rcv_rpl_long  : 0          avg_rcv_rpl_t : 0.0056
 rpl_rcv_count : 659        rel_rpl_rcv_t : 1.1518     abs_rpl_rcv_t : 1.1520
 dev_que_len_0 : 137        dev_que_len_1 : 12         dev_que_len>1 : 6
-------------------------------------------------------------------------------

nixnutzlinux:/opt/sdb/programs/bin # ./x_cons MAXDB1 SHOW T_C T45

SERVERDB: MAXDB1


--------------------  T45   User      ( pid =      0 ) -----------------------
 dispatcher_cnt: 45122                                 command_cnt   : 2302
 exclusive_cnt : 12794233                              self_susp_cnt : 41886
 Resume count 0  total 275        History [ T12 T4 T2 ]
 self_read_io  : 4300                                  avg_self_rd_tm: 0.0000
 self_write_io : 2419       self_write_pg : 2419       avg_self_wr_tm: 0.0006
 dev_read_io   : 492        rel_dev_rd_tm : 0.0006     abs_dev_rd_tm : 0.0006
 dev_read_pg   : 492                                   pages_per_io  : 1.0
 dev_write_io  : 92         rel_dev_wr_tm : 0.0020     abs_dev_wr_tm : 0.0021
 dev_write_pg  : 92                                    pages_per_io  : 1.0
 state_vsusp   : 2          rel_susp_time : 0.1458     abs_susp_time : 0.1458
 rcv_rpl_count : 692        rcv_rpl_long  : 2          avg_rcv_rpl_t : 0.0994
 rpl_rcv_count : 691        rel_rpl_rcv_t : 1.1367     abs_rpl_rcv_t : 1.1369
 dev_que_len_0 : 536        dev_que_len_1 : 39         dev_que_len>1 : 9
-------------------------------------------------------------------------------
```

As said, x_cons is a mighty tool. If you're a SAP R/3 customer, then the MaxDB CCMS and DB50 do a very good job hiding most of the complexity from you. If you're not, check it Database Manager GUI page Check => Database Server before you go down to the level of x_cons. So far there's no comparable application for non SAP customers.

The Diagnose Monitor

Compared to x_cons, the Diagnose Monitor is very easy to understand and even more useful for your daily maintenance work. The DIAGNOSE MONITOR is used to scan for critical and expensive queries. When turned on all queries that exceed a specified runtime, access more than a specified number of pages or suffer from low selectivity are logged.

The DIAGNOSE MONITOR stores it's result in three system tables: SYSMONITOR, SYSPARSEID, SYSMONDATA. The system tables should be cleared with the following SQL statement before you turn on the DIAGNOSE MONITOR in order to remove data collected during previous runs.

```
DIAGNOSE MONITOR CLEAR
```

To avoid confusion with settings from a previous run you should continue to configure the Diagnose Monitor before you turn it on. The figures collected by the monitor are written to three system tables. The tables are overwritten cyclically to prevent the database from overflowing. MaxDB makes the assumption that your database applications have a more or less fixed number of different queries and there is no need to record more than this fixed and limited number of queries. By default a maximum of 255 queries are stored. You should fine-tune this value to fit your needs. For large applications the default of 255 is too low, it would remove entries too soon from the list. To be on the safe side can use the ROWNO command to change the size of the que to the largest possible value which is 3.000.

```
DIAGNOSE MONITOR ROWNO 3000
```

Once you've decided how many rows should be recorded you need to tell MaxDB which parts need to be logged. The MaxDB parser splits every query into two parts. If you have a query like SELECT col1 FROM t1 WHERE col1 = 1, then MaxDB splits it into SELECT col1 FROM t1 and col1 = 1. The first part contains the SQL upto the first parameter. The second part are the parameters. Whenever MaxDB executes the same query with different parameters in the same session it does not parse the first part of the query again but only evaluates the parameters. In contrast to some other major databases MaxDB does always re-evaluate the parameters to get the best execution plan from the cost-based optimizer. However, check the documentation on the Shared SQL feature of 7.6 to learn more about the SQL parsing. For the sake of the Diagnose Monitor it's sufficient to know that the split exists and that you need to enable the logging of parameters to the SYSMONDATA table manually.

```
DIAGNOSE MONITOR DATA ON
```

The preparations are done after CLEAR, ROWNO and DATA ON. You can start to configure the more interesting threshold values that determine if a query is logged or not. The three thresholds that can be defined are: TIME, SELECTIVITY and READ. The TIME threshold is measured in milliseconds. A good-practice starting point is a value of 5000 (5 seconds). Like with all hints on starting points you need to adjust the value with trial-and-error to meet your requirements. If it collects two many queries, your queue flows over and queries get replaced in the queue before you can analyze them, you should increase the value. If not a single query is recorded you should lower the value.

```
DIAGNOSE MONITOR TIME 5000
```

Not only long running queries but also queries with a low selectivity need your attention. The selectivity is defined as the ration of the number of qualified reads divided by the number of total reads. Queries that do not use indexes but table scans to find the requested records do have a low selectivity. For example, it might be necessary to read and investigate 1000 records in order to find one matching row which gives you a selectivity as low as 0,1 percent. The SELECTIVITY threshold is measured in per-mill. Use 10.000 (10 percent) as a starting point.

```
DIAGNOSE MONITOR SELECTIVITY 10000
```

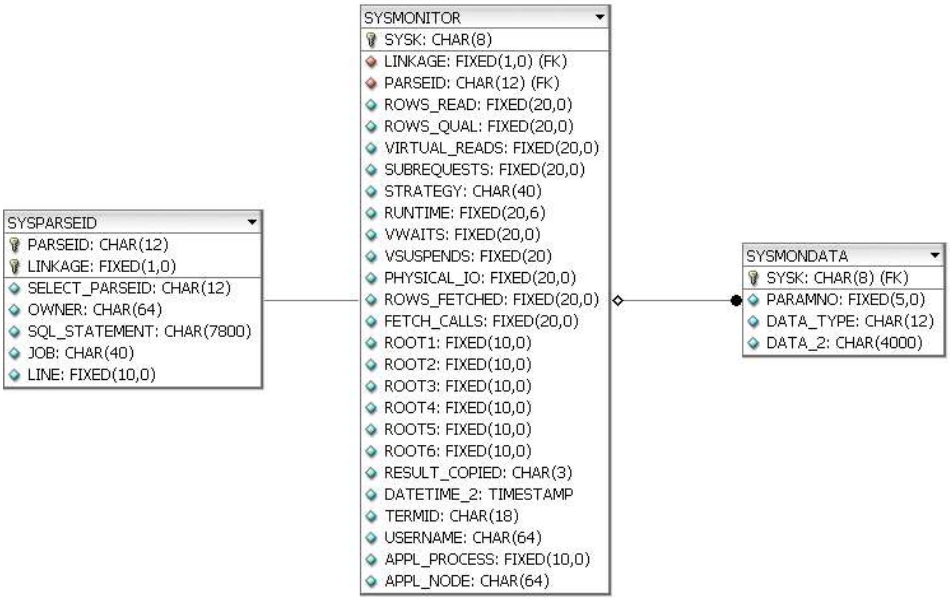
The last group of statements that can be found by the help of the Diagnose Monitor are statements that read many pages. Many page reads can – but must not due to the data cache! - indicate high I/O load. But even if the page reads do not map directly to I/O operations they cause a lot of work for the database kernel. Try a setting of 100 for your first tests. 100 pages with a page size of 8kb are equal to 800kb that need to be read during the query processing.

```
DIAGNOSE MONITOR READ 100
```

That's it! The Diagnose Monitor has been properly configured and turned on with the command sequence: CLEAR, ROWNO 3000, DATA ON, TIME 5000, SELECTIVITY 10000 and READ 100. To turn off the monitoring use:

```
DIAGNOSE MONITOR OFF
```

Sadly, there's no graphical tool to check the results of the monitoring unless you have access to a DB50 or the MaxDB CCMS inside a SAP system. The monitoring results are stored in three system tables: SYSPARSEID, SYSMONITOR and SYSMONDATA. The SYSPARSEID table contains all queries that are recorded while the Diagnose Monitor is on. The SYSMONITOR table gets cyclically overwritten if it contains more rows than specified with ROWNO. SYMONDATA records the the parameters of the SQL commands if available.



The ER diagram shows how the three tables are connected. Selecting the data of individual statements is not too complicated. The most important figures for an individual query are the selectivity, the runtime and it's physical_io usage. If you're unsure what the columns mean, check the improved documentation of the existing system tables or use the SQL Studio to view the table definitions and check the comments that are available for the columns.

```
SELECT
  sysmonitor.parseid,
  rows_read,
  rows_qual,
  round((100/rows_read)*rows_qual, 4) as selectivity_percent,
  virtual_reads,
  subrequests,
  strategy,
  runtime,
  vwaits,
  vsuspends,
  physical_io,
  rows_fetched,
  fetch_calls,
  result_copied,
  datetime,
  username,
  sql_statement
FROM
  sysmonitor,
  sysparseid
WHERE
  sysmonitor.parseid = sysparseid.parseid
ORDER BY
  runtime
DESC
```

If necessary you can query the SYSMONDATA table to get a list of parameters associated with a query that you want to analyze in more detail. Simply join the SYSMONDATA table and the SYSMONITOR table on SYSMONITOR.SYSK = SYSMONDATA.SYSK. Try the following statement for that purpose. There's no reason for using SUBSTR() in the query but to avoid the error -2003 Output columns too long.

```
SELECT
  sql_statement,
  paramno,
  data_type,
  SUBSTR(sysmondata.data, 1, 123) AS paramdata
FROM
  sysparseid,
  sysmonitor,
  sysmondata
```

```
WHERE
  sysparseid.parseid = sysmonitor.parseid AND
  sysmonitor.sysk = sysmondata.sysk AND
  sysparseid.parseid = <id_of_your_query>
ORDER BY
  paramno ASC
```

Even without the GUI a log with expensive queries is a rich set of information and a good starting point for collecting queries that need to be examined with EXPLAIN for further optimization. We recommend writing a script that collects informations from the monitoring tables and runs EXPLAIN on the SQL statements that have been logged. Note that the EXPLAIN you'll see when you re-execute the queries could show a totally different execution plan to the one used during the first execution of the query! The optimizer statistics might have been updated, large parts of the underlying data could have changed or new indexes could have been created meanwhile.

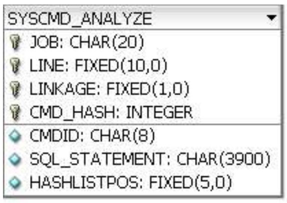
We've discussed the basics of the tool and you should be able to use it for your daily work. Some users of the MySQL server might wonder where the -log-queries-not-using-indexes is hidden in the MaxDB feature set. The SELECTIVITY command is a comparable and more powerful alternative. Indexes are additional search structures that reduce the number of pages that need to be read to find the requested records. The ratio of all pages read compared to the number of pages that contain requested records is the selectivity. As a rule of thumb one can say that if an index gets used the selectivity is high. A high selectivity means that few pages need to be read to find the requested records.

So far we showed how to to check individual queries. The Diagnose Monitor records every query that exceeds certain threshold values. You don't know if a slow query has been executed only once and a general load-peak has caused the slowness or if a slow query gets executed very often, it's really slow and it's the cause of the high load.

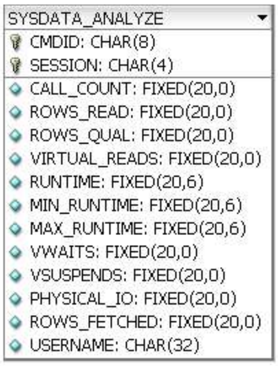
Aggregating values collected by the Diagnose Monitor is a tricky issue. And it's quite often not what you want. Remember that after ROWNO records MaxDB starts to overwrite data that's been collected. So you are limited to aggregating values for a certain time frame. The time frame that's needed to fill your log queue. You don't know if you got the time frame that's typical for your application or if you got a untypical time frame. Here comes the solution.

The Resource Monitor

DIAGNOSE ANALYSE is the tool for aggregated resource monitoring of SQL statements. Unlike the Diagnose Monitor only one entry in the command list is generated for a every query. If the same query gets executed again in the same session no new entry in the list of commands is created. But the resource consumption is recorded and added to aggregated resource consumption values in the resource list.



SYSCMD_ANALYZE
JOB: CHAR(20)
LINE: FIXED(10,0)
LINKAGE: FIXED(1,0)
CMD_HASH: INTEGER
CMDID: CHAR(8)
SQL_STATEMENT: CHAR(3900)
HASHLISTPOS: FIXED(5,0)



SYSDATA_ANALYZE
CMDID: CHAR(8)
SESSION: CHAR(4)
CALL_COUNT: FIXED(20,0)
ROWS_READ: FIXED(20,0)
ROWS_QUAL: FIXED(20,0)
VIRTUAL_READS: FIXED(20,0)
RUNTIME: FIXED(20,6)
MIN_RUNTIME: FIXED(20,6)
MAX_RUNTIME: FIXED(20,6)
VWAITS: FIXED(20,0)
VSUSPENDS: FIXED(20,0)
PHYSICAL_IO: FIXED(20,0)
ROWS_FETCHED: FIXED(20,0)
USERNAME: CHAR(32)

The list of SQL statements is stored in the SYSCMD_ANALYZE table and the aggregated resource figures are written to the SYSDATA_ANALYZE table. Like with the Diagnose Monitor one should clear the tables before the resource monitor is turned on. If you have the necessary permission this can be done by a DELETE statement but generally speaking one should never modify the content of any system internal tables if there's a special command for doing it. Like a developer should never access the internals of the module of another developer you should also follow the principles of encapsulation and decoupling and use the following commands to clear the tables. CLEAR DATA removes all resource consumption entries from the table SYSDATA_ANALYZE, CLEAR COMMAND removes all entries from the table SYSCMD_ANALYZE and CLEAR ALL clears both tables.

DIAGNOSE ANALYSE CLEAR DATA

DIAGNOSE ANALYSE CLEAR COMMAND
DIAGNOSE ANALYSE CLEAR ALL

To turn on the resource monitoring two steps are needed. The first command turns on the recording for all commands. The second step turns on the recording of the resource consumption. Further configuration is not necessary. The resource monitor simply traces all commands in the current session and in all sessions that are opened from now on.

DIAGNOSE ANALYSE ON
DIAGNOSE ANALYSE COUNT ON

The monitoring can be turned off with the following commands.

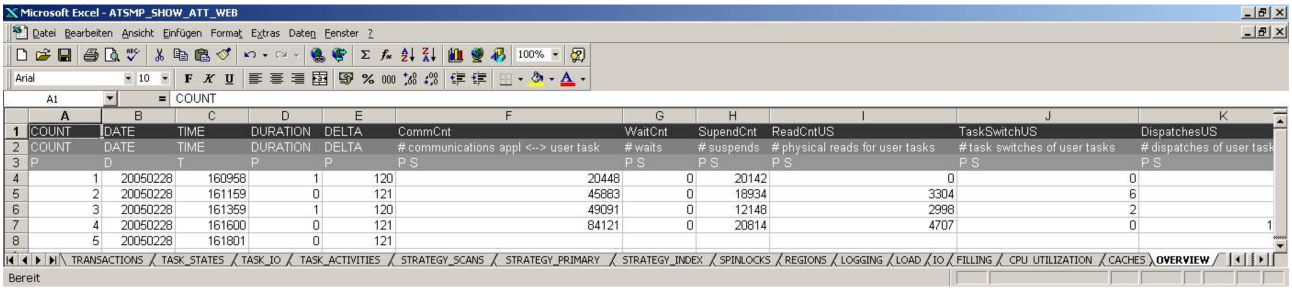
DIAGNOSE ANALYSE COUNT OFF
DIAGNOSE ANALYSE OFF

We are sorry to say but for non-SAP users there's no graphical tool to browse the monitoring results. However the lack of a graphical tool should not be a major problem. Try the following query to get a list of queries sorted by the total runtime. Once you have such a list it's up to you to do the further analysis! No GUI can take this task from you.

```
SELECT
  sql_statement,
  session,
  call_count,
  rows_read,
  rows_qual,
  ROUND((100/rows_read) * rows_qual, 4) AS selectivity_percent,
  virtual_reads,
  runtime,
  min_runtime,
  max_runtime,
  runtime/call_count AS avg_runtime,
  vwaits,
  vsuspends,
  physical_io,
  rows_fetched,
  username
FROM
  syscmd_analyze as sc,
  sysdata_analyze as sd
WHERE
 sc.cmdid = sd.cmdid
ORDER BY
 runtime DESC
```

Statistics of the Database Analyzer

The last performance analysis tool that's covered in this introduction is the Database Analyzer. The Database Analyzer is used to periodically collect informations about the health status of the database. About 180 individual figures are recorded and grouped into 16 sets. The record format is CSV, comma-separated values and thus it can be easily imported into Excel or other tools for post processing



	A	B	C	D	E	F	G	H	I	J	K
1	COUNT	DATE	TIME	DURATION	DELTA	CommCnt	WaitCnt	SupendCnt	ReadCntUS	TaskSwitchUS	DispatchesUS
2	COUNT	DATE	TIME	DURATION	DELTA	# communications appl <--> user task	# waits	# suspends	# physical reads for user tasks	# task switches of user tasks	# dispatches of user task
3	P	D	T	P	P	P S	P S	P S	P S	P S	P S
4	1	20050228	160958	1	120	20448	0	20142	0	0	
5	2	20050228	161159	0	121	45883	0	18934	3304	6	
6	3	20050228	161359	1	120	49091	0	12148	2998	2	
7	4	20050228	161600	0	121	84121	0	20814	4707	0	1
8	5	20050228	161801	0	121						

The Database Analyzer can be turned on and off if the database is in online mode. The easiest way to turn it on is to use the Database Manager GUI. Select Check -> Database Analyzer and answer the questions in the start dialog. Select an interval for recording of figures. We recommend 10 seconds for short-time and 900 seconds for long-time observations. Choose then how often the metering shall be done. A value 0 or no value turns it on until the Database Analyzer gets turned off. Unless you're using MaxDB in a SAP environment, you do not need to specify a configuration file. The configuration file contains threshold values that assign warning levels to recorded figures if required. If a value exceeds a threshold a warning it is written to the file DBAN.prt file. The following shows an excerpt from the DBAN.prt log file.

```
===== #1          at 2004-11-30 14:16:27
*  I  SQL commands executed: 100
      CON: PureSQL_Cmds > INTERVAL * 5
      VAL: 100          > 5        * 5
* W3  Catalog cache hitrate (SQL Pages) 73.40%, 832 of 3128 accesses failed
      CON: Cat_Hit < 80 && ( SQL_Cmds ) > INTERVAL
      VAL: 73.40   < 80 && ( 350      ) > 5
```

Use the default dbanalyzer.cfg configuration file for your first steps. It's contained in the directory INSTROOT/env/ . In oder to use it, simply leave out the field “Configuration File” in the start dialog of the Database Analyzer. The last two options you have in the start dialog are a connect-on-restart option and a checkbox to remove all previously recorded values of the current day. If you prefer to use a command line program to start the Database Analyzer, run the dbanalyzer command line utility.

```
# /opt/sdb/programs/bin/dbanalyzer
MaxDB Database Analyzer, The Performance Analysis Tool, Version 7.5.00.19
Copyright 2000-2004 by SAP AG

Enter database name: MAXDB1
Enter user name: DBA
Enter password:

Used protocol directory: /var/opt/sdb/data/wrk/MAXDB1/analyzer
Used configuration file: /opt/sdb/programs/env/dbanalyzer75.cfg
```

Once you've turned the Database Analyzer on, the Database Analyzer starts to write values into 17 files. Unless you've specified a different location the files are created in the directory RUNDIRECTORY/analyzer/YYYYMMDD/. Again, the easiest way to work with them is to use the Database Manager GUI. Select Check -> Diagnosis Files -> DB Analyzer File [FOLDER] and the folder for the current day.

In most cases you don't need to worry about slowdowns caused by the additional load that the Database Analyzer puts on the system. First the extra load is negligible and second you should learn that SAP is turning it on by default for R/3 systems. The benefits you can get from it clearly outweigh any theoretical drawbacks.

The DBAN.prt file contains an entry for each monitoring interval. The entries will either show OK to indicate that nothing special has happened during the time period or it will show warnings and informations. Informations are marked with an “I” and can be more or less ignored. Warnings indicate problems. Warnings are divided into three levels. Warning 1 (W1) is the least critical warning whereas Warning 3 (W3) are serious warnings that should be followed by an user action to lower the number of critical warnings.

While some warnings which are derived from a total of about 160 measured figures are easy to understand others are not. For example the group FILLING that gives fill level informations for data- and logvolumes can be understood by beginners. On the other hand the SPINLOCKS group shows informations which most people won't even understand if you tell them what a spinlock is. However, we'll leave this topic for following articles.

Conclusion

The intention of this article was to give you an overview of performance analysis related tools that come with MaxDB. We said in the introduction that this article will not be sufficient to teach you all the details. In fact it's quite often limited to telling you that a certain tool exists and giving you instructions on how to turn it on. Be patient and get back to us for the next article. We can't promise a release date but we might soon start to write a second article that concentrates on the details.

If you can't wait that long, visit us on the MaxDB mailinglist or meet us in real life during a training class.