

# Managing DB2 Performance Across Linux, Unix, Windows and z/OS

---

## **White Paper**

*written by  
Jim Wankowski,  
DB2 Technology Specialist,  
Quest Software, Inc.*



**© 2006 Quest Software, Inc. All rights reserved.**

The information in this publication is furnished for information use only, does not constitute a commitment from Quest Software Inc. of any features or functions discussed and is subject to change without notice. Quest Software, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this publication.

Last revised: July 2006

# TABLE OF CONTENTS

<b>INTRODUCTION</b> .....	<b>4</b>
<b>DEFINING PERFORMANCE</b> .....	<b>4</b>
<b>MONITORING</b> .....	<b>4</b>
<b>MONITORING METHODS</b> .....	<b>5</b>
z/OS .....	5
LUW .....	6
<i>Snapshot Monitor</i> .....	6
<i>Event Monitor</i> .....	6
<b>MEMORY MANAGEMENT</b> .....	<b>7</b>
MEMORY USAGE .....	7
<i>EDM Pool</i> .....	7
<i>Catalog Cache/Package Cache</i> .....	8
<i>Sorting</i> .....	8
<i>RID Pool</i> .....	9
<i>Bufferpools</i> .....	10
<b>LOCKLIST</b> .....	<b>12</b>
PERFORMANCE IMPLICATION .....	12
WHAT TO MONITOR.....	12
MINIMIZING LOCK USAGE.....	12
<b>SELF-TUNING MEMORY</b> .....	<b>13</b>
<b>PHYSICAL DESIGN</b> .....	<b>14</b>
TABLESPACES.....	14
z/OS.....	14
LUW.....	14
INDEXES .....	15
<i>Managing Space</i> .....	15
<b>MAINTENANCE</b> .....	<b>16</b>
REORGANIZATION .....	16
<i>What to Monitor</i> .....	16
STATISTICS COLLECTION.....	17
<b>APPLICATION DESIGN</b> .....	<b>18</b>
OPTIMIZATION .....	18
z/OS.....	18
LUW.....	18
<i>Optimization Tips</i> .....	19
<b>SUMMARY</b> .....	<b>20</b>
<b>ABOUT THE AUTHOR</b> .....	<b>21</b>
<b>ABOUT QUEST SOFTWARE, INC.</b> .....	<b>22</b>
CONTACTING QUEST SOFTWARE .....	22
TRADEMARKS.....	22

# INTRODUCTION

Obtaining optimal performance from your DB2 applications is a multi-faceted exercise. DB2 performance optimization cannot be limited to simply tuning SQL. You need to have an understanding of how factors such as memory, physical design, and maintenance can affect performance. If you are a DBA having to manage DB2 on multiple platforms, it is important that you have an understanding of what needs to be monitored.

This whitepaper will discuss how to identify and resolve the most common areas of performance bottlenecks in DB2 running on Linux, Unix, Windows (LUW) and z/OS. In addition this whitepaper will compare/contrast memory architecture, physical design, maintenance and SQL tuning techniques for applications running in z/OS vs. Linux, Unix or Windows.

## DEFINING PERFORMANCE

The Merriam-Webster Dictionary defines performance as “The fulfillment of a claim, promise or request.” Performance can mean different things to different companies. Some companies may define performance as 24x7x365 availability, while others may have specific transaction throughput requirements or minimum response times. Different tuning techniques need to be applied to these different scenarios.

## MONITORING

In order to tune and maintain a subsystem or instance, one must regularly monitor the entire DB2 environment. Too many people mistakenly focus all of their efforts on tuning SQL and do not look at the big picture. Think of your DB2 installation as an ecosystem, where memory, physical design and SQL transactions all work in conjunction with each other, and all can have a direct impact on performance. Proper SQL coding is probably the most critical factor related to performance, but it is very important to understand that the best written SQL statements will not perform properly if there are inadequate memory allocations or a poor physical database design.

# MONITORING METHODS

Whether you are monitoring a z/OS subsystem or an LUW instance, it is important to understand the monitoring methods that are available to you.

## z/OS

z/OS uses the instrumentation facility, which has five classes of traces:

1. Statistics
2. Accounting
3. Performance
4. Monitor
5. Audit

The monitor classes are activated with the “-START TRACE” command. The most commonly used traces are the statistics and accounting classes. The statistics class gives you information at a global level, summarizing total activity within a subsystem. The accounting class traces give you detailed performance metrics at the individual thread level. The accounting class traces give you a good level of detail for monitoring specific applications. Both classes have relatively low overhead and are typically turned on all the time. You can typically diagnose 80-90 percent of your performance problems using the information provided with these traces. The monitor class trace is used to externalize this information to make it available to performance monitors or homegrown applications.

When a server performance problem occurs and there is a need for greater detail, the performance class traces can be used. These must be used with caution because they can incur extreme overhead. It is best to qualify the trace as much as possible to limit the scope of information collected.

The final trace type is the audit trace. As the name implies, it is used for auditing access to sensitive data. It does not provide any performance-related data. The table DDL has to be physically altered to include the audit keyword, then the audit traces can be started for specific tables to monitor read/write/update activity against the tables.

# LUW

When monitoring a DB2 instance running on Linux, Unix or Windows, you have two monitoring choices: the Snapshot monitor or the Event monitor. For both types of monitors you have the choice of storing the collected information into a file or a DB2 table.

## Snapshot Monitor

As the name implies, the Snapshot monitor takes a “snapshot” of activity at a specific point in time. So, whether a transaction is actually active at this point in time will determine if performance information will be collected. The “Get Snapshot” command is issued to execute the snapshot. The snapshot can be qualified to retrieve only specific data, such as locking, bufferpool usage, etc.

The Snapshot monitor provides a great level of detailed performance information at a relatively low overhead. It is most useful in diagnosing immediate performance problems but also can be useful in doing trend analysis, by issuing snapshots at regular intervals.

## Event Monitor

Event monitors provide the ability to collect activity over a period of time. Event monitors are activated with the “CREATE EVENT MONITOR” SQL and collect specific “events,” such as deadlocking, connection activity, SQL statements, etc., over a historical period. This provides the ability to do workload analysis for tracking trends and problematic transactions.

# MEMORY MANAGEMENT

Minimizing physical I/O is one of the most important aspects of performance tuning. Proper allocation of memory is key to ensuring optimal performance. The best written SQL statement will perform poorly if there is inadequate sort, buffer or system memory available.

Memory usage, however, requires monitoring at regular intervals. DB2 environments tend to change as a result of increases in data volumes, transaction rates, numbers of users, etc. Therefore, you cannot take a “set it and forget it” approach to memory allocation. The following table illustrates the memory areas that can directly impact application performance. This section compares and contrast these memory areas, discusses how they can impact performance, as well as examines monitoring and tuning techniques.

Z/OS	LUW
EDM Pool	Catalog Cache Package Cache
RID Pool	N/A
Sort Pool	Sort Heap
Bufferpool	Bufferpool
N/A	Locklist

Fig. 1 Comparable memory areas

## Memory Usage

### EDM Pool

The Environmental Descriptor Manager pool, or EDM pool, can be described as the System Bufferpool. It is used whenever an application is loaded for execution. The EDM pool contains the following information:

- Database Descriptor (DBD) information
- CURSOR Table (CT) information
- PACKAGE Table (PT) information
- Skeleton Cursor Tables (SKCT)
- Skeleton Package Tables (SKPT)
- Plan/Package authorization cache
- Dynamic SQL cache

## **Performance Implications**

Possible performance implications of an inadequate EDM pool allocation are an increase in physical I/O against DSNDB01; a slowdown in application loading time due to the reloading of SKCT, SKPT and DBD information; excessive SQL dynamic re-preparation (if dynamic SQL caching is on); a decrease in thread concurrency; and, in the worst case, a complete failure due to resource unavailable condition (SQL code of -904).

## **Catalog Cache/Package Cache**

Comparable in purpose on the LUW side are the catalog cache and the package cache. The catalog cache is used mainly in program preparation, so it is most critical to monitor when a lot of development is occurring. The catalog cache minimizes I/O against the catalog when it obtains object information from SYSTABLES, authorization checking from SYSDBAUTH, and execute privileges for routines. The package cache is used for loading packages and caching dynamic SQL.

### **Performance Implications**

Possible performance implications for inadequate catalog cache are an increase in compile/bind times and increased time to check database and execution privileges. If your environment is static, with little or no application development, you should consider minimizing the memory allocated to the catalog cache and utilizing it somewhere else, such as bufferpools. An inadequately sized package cache will result in a slowdown in dynamic SQL execution due to excessive re-preparation, as well as a slowdown in loading packages.

### **What to Monitor**

For EDM pools, you want to maintain a hit ratio of at least 80 percent. Package/cursor table hit ratios should be 80–90 percent, and DBD requests should be as close to 100 percent as possible.

The hit ratio for package and catalog caches should be within 80–90 percent. The Snapshot monitor will also provide overflows for these caching areas, which is a red flag for insufficient allocation.

## **Sorting**

Efficient sorting is critical to SQL performance. You want to avoid physical disk I/O by doing as much sorting in memory as possible.

### **Sort Pool**

The sort pool, the memory area where all sorting occurs, works in conjunction with logical work files residing in tablespaces located in the DSNDB07 database. Ideally the sort should occur in memory without any physical I/Os, but, with today's large data volumes, this is unlikely.

## Sort Heap

Similar to the sort pool of z/OS, this is the memory area for all necessary sort activity. The sorting can be overflowed into a temporary table within the database if inadequate memory is available. This is a similar concept to the DSNDB07 sort files in z/OS.

## Optimizing Sort

There are certain things you can do to optimize sorting within these memory areas. First, the sort area should be made as large as possible. For z/OS, you can optimize your logical work files in DSNDB07 by allocating four to five equally sized tablespaces, each using their own bufferpools. Set the bufferpools' sequential steal threshold to 100 percent.

Also, try to separate the tablespaces into different volumes, keeping the allocation in primary space. Indexing is the best option for minimizing sort on all platforms. You will want to adhere to simple rules of thumb, such as avoiding ORDERBYs, GROUPBYs or DISTINCTS.

Also, you will want to avoid sorting VARCHAR fields. Remember that these fields have to be expanded to their full size during a sort, thereby using a lot more memory. Finally, try to select only the columns needed in your SQL statements. Remember that all column data has to be loaded into the sort area, not just the columns in the order by clause.

## What to Monitor

For LUW sort heaps, you want to monitor sort heap overflow and sort heap threshold. An overflow within sort heap will result in a non-piped sort vs. a piped sort. A non-piped sort means that there is insufficient memory to hold the entire sort result set, so the sorting has to be externalized to temporary tables in the database, then merged together for the result set.

## RID Pool

The "Row Identifier Pool" is used for sorting index RIDs for list prefetch, multiple index access paths or hybrid joins.

## Performance Implication

An inadequate RID pool can cause a RID pool overflow. As a result, transactions that normally would use list prefetch, multiple index access or hybrid joins instead revert to a tablespace scan.

## What to Monitor

There are three main things to monitor for RID pools:

1. Insufficient pool size, which will result in RID pool failures that will cause access paths to revert to tablespace scans.
2. RDS Limit – This is when the optimizer determines that the results set will be more than 25 percent of the total rows in the table. The optimizer will then choose a tablespace scan over an index scan. If this condition occurs, be sure that the catalog statistics are accurate.
3. DM limit – The Data Manager limit is encountered when the total number of RIDs to be sorted exceeds 16 million. This is a DB2 limit, and a tablespace scan will be performed. This is an extreme condition that might occur, but only in very large environments. If a query is producing such a large results set, a tablespace scan may be the best option. You can consider trying to re-evaluate the indexes on the table(s), or possibly adding additional filtering to reduce the results set.

## Bufferpools

Bufferpools are areas of virtual storage where DB2 maintains data pages to satisfy queries without having to do physical I/O to the DB2 tables. The goal is to keep as many frequently accessed data pages in memory as possible. Bufferpools are by far the most critical memory area when it comes to performance for all platforms of DB2. A tremendous boost in performance can be obtained with properly tuned bufferpools.

### z/OS

In z/OS v8, you have the option of allocating up to 50 4K bufferpools, and 10 8K, 16K and 32K bufferpools. At a minimum, you have to allocate one 4K buffer (BP0), one 8K (BP8K0), one 16K (BP16K0) and one 32K buffer (BP32K). The z/OS bufferpools have several customizable thresholds that allow you to tailor a bufferpool to a specific application, such as random access vs. sequential access, frequent updates, etc. Some companies actually change their bufferpool settings throughout the day to adjust for process changes. Refer to IBM's *DB2 Administration Guide* for details on how to tune these parameters.

### LUW

An LUW database must have at least one bufferpool. A default bufferpool (IBMDEFAULTBP) is created automatically when a new database is created. A series of hidden bufferpools are also created in 4K, 8K, 16K and 32K page sizes. These are there in case a normal bufferpool of the required page size is unavailable due to insufficient memory, or the normal bufferpool is not active for some reason. These hidden bufferpools do not appear in the system catalog or bufferpool system files. A new auto-tune feature in v9 will allow DB2 to automatically size your bufferpools based on transaction load. This will take a lot of the guesswork out of tuning your bufferpools.

## Performance Implication

The general performance implication for inadequately sized or inefficiently used bufferpools is an increase in physical I/O due to reading data pages and having to flush pages out of bufferpools that are too small. Both situations can severely impact performance. In an LUW environment, performance will degrade substantially if DB2 needs to use the hidden bufferpools. A warning message will be written to the administration notification log if this occurs.

## Effective Use of Bufferpools

Bufferpool tuning is somewhat of an exercise in trial and error. Every DB2 shop has different types of databases and transactions that require their own unique bufferpool implementation; however, there are some rules of thumb. The old z/OS rule was to just throw everything into BP0 and not worry about it. Today, we know that this is the worst possible performance mistake you can make! BP0 should contain only your catalog and directory. As of DB2 v8, we can allocate up to 80 bufferpools. IBM did this for a reason. Use the guidelines in the following table as a starting point for defining/tuning your bufferpools.

<b>Z/OS</b>	<b>LUW</b>
Catalog/directory – BP0	System catalog
DSNDB07	Sequentially scanned tables
Tablespaces	Temporary tablespaces
Indexspaces	Small frequently updated tables
Small read-only tables	Small read-only tables
Large TSs with random access	Large tables w/random access
Small, frequently updated tables and IXs	
Test environment	

**Fig. 2 Bufferpool guidelines**

## What to Monitor

For both z/OS and LUW you want to monitor the bufferpool hit ratios as well as paging. Excessive paging implies that the bufferpools are not large enough and that pages are constantly being flushed. The key to monitoring bufferpools is to make changes, then monitor over a period of time.

# LOCKLIST

The locklist controls the amount of memory available for managing locks across all concurrently running applications. There is one locklist per database in DB2 LUW. The other parameter that works in conjunction with the locklist is *maxlocks*. *Maxlocks* defines the percentage of the locklist that an application must be using before lock escalation occurs.

## Performance Implication

When inadequate memory is available for the locklist, lock escalation can occur, going from a row lock to a table lock. This, in turn, can result in performance degradation due to a decrease in application concurrency caused by lock waits and potential deadlocks.

## What to Monitor

Monitoring lock escalations is essential. If there are frequent lock escalations, you will need to adjust the locklist and *maxlocks* values. The locklist should also be increased if the *maxappls* parameter is increased. *Maxappls* defines the number of applications that can run concurrently.

## Minimizing Lock Usage

Following are some ways to minimize locking:

- Use frequent COMMITs during updating.
- Consider using LOCK TABLE for applications performing large numbers of updates.
- Use CURSOR STABILITY.
- Set locklist to AUTOMATIC (v9).

## SELF-TUNING MEMORY

DB2 UDB for LUW v9 introduced the concept of self-tuning memory. The feature simplifies the task of managing package cache, sort heap, bufferpools and the locklist. By specifying the *database\_memory* parameter as *AUTOMATIC*, DB2 will dynamically adjust these memory parameters based on database workload. This parameter can be left on, or, once a typical workload has been set, the values can be frozen by turning off the automatic parameter.

# PHYSICAL DESIGN

The physical design of a database can definitely impact application performance. The way tablespaces are defined can determine how the data will be accessed, and, of course, proper index design is essential for SQL to achieve peak performance.

## Tablespaces

### z/OS

When designing new databases in z/OS, you can choose from four types of tablespaces: simple, segmented, partitioned and large. As a rule, simple tablespaces should be avoided. Segmented tablespaces are best for most applications, since they have much more efficient I/O than a simple tablespace for either single or multiple table per tablespace designs.

Partitioned tables primarily provide manageability for very large volumes of data. Data can be broken into partitions based on key ranges, and each partition can have maintenance performed on it independently. This makes activities like reorgs, backups, RUNSTATS, etc., much more manageable. The second benefit of a partitioned tablespace is query parallelism, or the ability for a query to read data from multiple partitions simultaneously. Large tablespaces are used for managing LOB data such as BLOBs and CLOBs.

### LUW

The concept of simple, segmented or partitioned tablespaces does not exist in the LUW environment. You have the choice of regular, temporary or large. Regular tablespaces are similar in structure to segmented tablespaces in z/OS in that they are defined with extents, which are blocks of 2-256 pages. Partitioning is a completely different concept in the LUW world, as it only applies to environments that are partitioned via the data partitioning option of DB2. In this type of environment, the database itself is partitioned across multiple servers or nodes.

# Indexes

Proper index design is critical for obtaining optimal performance from your SQL. You need to have a thorough understanding of all transactions accessing a table in order to make accurate index decisions. Here are some general rules of thumb:

- Define indexes on foreign keys
- Define indexes that include all columns in a query if possible
  - Index only access
- Define indexes on join columns
- Index highly sorted columns
- Minimize indexes on highly updated tables

## Managing Space

Some guidelines should be observed when managing the physical space of your tablespaces. For z/OS, try to keep your VSAM datasets in mostly primary space. A high number of extents can cause excessive I/O. For LUW, you have the choice of allocating tablespaces as either System Managed Space (SMS), or Database Managed Space (DMS).

Advantages and disadvantages exist for both. When using an SMS you do not have to worry about specifying any space parameters. The tablespace uses the space available on the drive that it is allocated to. This is easier from an initial allocation perspective, but can be much more complex if you run out of space. You will have to run a redirected restore to move the file to another drive.

DMS is similar to z/OS in that you specify an allocation size, and the data files are physically allocated to that size. If you need to increase space, it's just a matter of running an alter command to add another container or resize the existing one. The other advantage to DMS over SMS is that you can separate your indexes into a separate tablespace. This will provide a performance advantage from an I/O perspective, as you will be avoiding any type of disk contention.

# MAINTENANCE

The two most important maintenance procedures that impact performance are data reorganization and statistics collection.

## Reorganization

Keeping your data physically organized is essential for the performance of both index and data scans. Disorganization of data occurs when data is inserted, updated, deleted and, for whatever reason, the data values cannot be returned to their original position, thereby becoming relocated. Performance will degrade due to an increase in I/O and getpages, if your data is poorly organized. If the cluster ratio falls below 80 percent, sequential prefetch is turned off, and access paths may convert to tablespace scans, which can have a severe impact on performance. DB2 catalog statistics will provide the information needed to determine when a reorganization is required.

### What to Monitor

#### z/OS

A sample job called DSNTESP in SDSNAMP contains sample queries to identify tablespaces that require reorganization. The statistics to monitor for tablespaces are:

- Cluster ratio < 95%
  - SYSINDEXES
  - SYSINDEXPART
    - a. Faroffpos > 10% OF card
- Excessive Row Relocation
  - SYSTABLEPART
    - a. NEARINDREF + FARINDREF > 10% CARD
- Excessive LEAF distance (Index reorg)
  - SYSINDEXPART
    - a. LEADDIST > 200
- Extents > 50
- Large Objects
  - SYSLOBSTATS
    - a. ORGRATIO > 2

## LUW

The REORGCHK command can be run to identify tables that require reorganization, or the following catalog statistics can be queried:

- Cluster Ratio < 90%
  - SYSCAT.INDEXES
- Overflow Rows
  - SYSSTAT.TABLES
    - a. OVERFLOW
- Fetch Statistics
  - SYSCAT.INDEXES
    - a. AVERAGE\_SEQUENCE\_FETCH\_PAGES
    - b. Increase in AVERAGE\_RANDOM\_FETCH\_PAGES
- Number of LEAF pages
  - SYSCAT.INDEXES
    - a. NLEAF

## Statistics Collection

Accurate statistics are essential for the optimizer to make intelligent access path decisions. They also are critical for helping DBAs monitor growth and assess the need for reorganizations. The RUNSTATS utility should be run after a LOAD, REORG or REBUILD INDEX, as well as whenever a new index is created or there is heavy update activity on a table. When the RUNSTATS utility is executed, you may or may not need to rebind applications.

Here are some general criteria for determining whether or not a rebind is needed after a RUNSTATS:

- A new index has been created
- The cluster ratio changes to greater than or less than 80 percent
- The cardinality changes by more than 20 percent
- Index LEAF distance changes by 20 percent

# APPLICATION DESIGN

Efficient application design is the single most critical component of performance. In fact, application design is a subject for an entire whitepaper. This section will discuss some of the general guidelines that developers should follow to maximize the performance of their SQL.

The main problem with SQL is its flexibility. A SQL statement can be written many different ways, with each way producing the same results set with different access paths and dramatically different elapsed times. Application developers need to have a good, fundamental understanding of SQL coding to ensure that inefficient SQL is not implemented into a production environment.

## Optimization

The DB2 optimizer determines the access path or “roadmap” to retrieve the desired results set. The optimizer looks at the way the SQL statement is coded, what indexes are available and the current catalog statistics to make its determination.

### z/OS

The z/OS optimizer is pretty much a fixed process. There are a few ways to influence the optimizer by either updating the catalog with simulated statistics or by using “Hints.” Hints help you manually update the access path information in the Plan\_Table. You can then assign the updated access path a name and specify the name in the bind DDL or in line for dynamic SQL.

### LUW

The LUW optimizer is much more flexible; it allows you to specify the level of optimization based on the complexity of the query. There are seven levels of optimization, and, the higher the level of optimization, the more resources that will be consumed. This is not a big factor with static applications because the optimization occurs only during bind time. However, in the case of dynamic SQL, much more care needs to be taken because the optimization will occur every time the SQL statement is prepared. This can result in unnecessary overhead due to excessive optimization. It should also be noted that a higher optimization level doesn’t necessarily translate into a better access path. The structure of the query needs to be taken into account in order to determine the best level.

## Optimization Tips

As stated earlier, it is essential to follow basic fundamentals in SQL coding to achieve the best performance. Following are some guidelines:

- Keep accurate statistics
- Keep predicates as restrictive as possible
- Only select required columns
- Avoid sorts
- Avoid UNION clauses
  - IN, BETWEEN or CASE is much better
- Use OPTIMIZE for N ROWS when possible
- Use FETCH FIRST N ROWS when possible
- Order predicates from most to least restrictive
- Use appropriate optimizer class (LUW)
- ***EXPLAIN!!!***

## SUMMARY

Good performance is truly a measure of customer satisfaction. When planning a performance tuning strategy, be sure to understand the business needs of the customer. Identify the most critical business applications and focus your tuning efforts on those first. And, remember that tuning is not a “set it and forget” process. Constantly monitor your critical applications for any type of environmental changes and adjust the applications accordingly.

## ABOUT THE AUTHOR

**Jim Wankowski** is currently the DB2 Technology Specialist at Quest Software. Jim has more than 20 years of development and database administration (DBA) experience with DB2. Jim participated in the original beta program for DB2 in 1984 and is well known in the DB2 community. In addition, he has written articles for *DB2 Magazine* and regularly presents at IDUG conferences, regional DB2 user groups and vendor seminars worldwide.

## ABOUT QUEST SOFTWARE, INC.

Quest Software, Inc. delivers innovative products that help organizations get more performance and productivity from their applications, databases and Windows infrastructure. Through a deep expertise in IT operations and a continued focus on what works best, Quest helps more than 18,000 customers worldwide meet higher expectations for enterprise IT. Quest Software can be found in offices around the globe and at [www.quest.com](http://www.quest.com).

## Contacting Quest Software

Mail:	Quest Software, Inc. World Headquarters 5 Polaris Way Aliso Viejo, CA 92656 USA
Web site	<a href="http://www.quest.com">www.quest.com</a>
Email:	<a href="mailto:info@quest.com">info@quest.com</a>
Phones:	1.800.306.9329 (Inside U.S.) 1.949.754.8000 (Outside U.S.)

Please refer to our Web site for regional and international office information. For more information on Quest Software solutions, visit [www.quest.com](http://www.quest.com).

## Trademarks

All trademarks and registered trademarks used in this guide are property of their respective owners.