



Best Practices Database Storage

Aamer Sachedina

*Senior Technical Staff Member
DB2 Technology Development*

Matthew Huras

DB2 LUW Kernel Chief Architect

Allan Risk

DB2 Information Development

Executive summary	3
Introduction to database storage	4
Goals of good database storage design.....	5
Simplicity in database storage design.....	5
Recipe for database storage success	6
Think about real physical disks, not just storage space	6
Have dedicated LUNs and file systems per non-DPF DB2 database server / per DPF partition	9
Stripe at most in two places.....	9
Separate DB2 transaction logs and data	10
Use file systems instead of raw devices—one file system per LUN.....	10
Where possible, use RAID-10 for transaction logs, RAID-10 or RAID-5 for data.....	10
Set DB2_PARALLEL_IO for optimal I/O parallelism.....	11
Use the NO FILE SYSTEM CACHING clause	12
Use DB2 automatic storage to stripe everything everywhere.....	12
Do not hand-tune the NUM_IOCLEANERS, NUM_IOSERVERS, and PREFETCHSIZE configuration parameters.....	14
Best Practices.....	15
Conclusion	16
Further reading.....	17
Contributors.....	17
Notices	18
Trademarks	19

Executive summary

In a world with networked and highly virtualized storage, database storage design can appear like a dauntingly complex task for a DBA or system architect to get right.

Poor database storage design can have a significant negative impact on a database server. CPUs are so much faster than physical disks that it is not uncommon to find poorly performing database servers that are significantly I/O bound and underperforming by many times their true potential.

The good news is that it is more important to *not get database storage design wrong* than it is to get it perfectly right. Trying to understand the innards of the storage stack and hand tuning which database tables and indexes should be stored on which part of what physical disk is an exercise that is neither generally achievable nor maintainable (by the average DBA) in today's virtualized storage world.

Simplicity is the key to ensuring good database storage design. The basics involve ensuring an adequate number of physical disks to keep the system from becoming I/O bound.

This document provides a recipe to a healthy database server through easy to follow best practices in database storage, including guidelines and recommendations for each of the following areas:

- Physical disks and logical unit numbers (LUNs)
- Stripe and striping
- Transaction logs and data
- File systems versus raw devices
- Redundant Array of Independent Disks (RAID) devices
- Registry variable and configuration parameter settings
- Automatic storage

Note: This paper covers best practices for deployments of DB2 for Linux, UNIX and Windows in typical OLTP environments. The recommendations discussed in this paper do not necessarily apply in data warehousing environments or in environments where a DB2 database is used as the underlying database for third-party software.

Introduction to database storage

Storage Area Networks (SAN) and Network Attached Storage (NAS) have fundamentally changed the database storage world. A decade or so ago, the word “disk” referred to a physical disks with heads and platters. In today’s storage world, a “disk” is a completely virtual entity that resides somewhere on the storage network and that can be either a single physical disk, a portion of a physical disk, a RAID array, or some combination of RAID arrays.

Recent enhancements in file systems, such as direct and concurrent I/O, have almost eliminated any performance advantage that using raw devices had over using file systems.

While Moore’s Law has held for CPU processing power, it does not apply to storage subsystem speeds. Despite changes in storage communication introduced by SAN and NAS, the underlying infrastructure of how bits are stored remains fundamentally the same—mechanical spindles rotate multiple platters of magnetic material, upon which are encoded bits of information.

While spindle speeds have increased, and caching of data on storage controllers using DRAM and NVRAM has helped, neither of these advancements have kept up to the sheer magnitude of processing power increases over the past decade or so. The result is that relative to CPU processing speeds, disks have become much slower. This speed difference requires an increasingly larger number of physical disks per CPU core to ensure that the system is not I/O bound. Given that platter capacities, which govern the actual capacity of each physical disk, have increased substantially, achieving an appropriate ratio of physical disks per CPU core remains a difficult feat to achieve.

Given the changes in storage, file systems, and CPU processing speeds, the best practices for database storage provisioning and management have also evolved. In the past, a DBA might have been advised to determine exactly on which physical disk and on which part of each physical disk individual tables and indexes should be placed. In today’s virtualized storage world, for the average DBA, yesterday’s best practices are impractical.

The best practices presented in this document have been developed with the reality of today’s storage systems in mind.

Refer to the “Best Practices for Physical Database Design” white paper for related information on database performance and speed of database operations. This white paper and others are available at the DB2 Best Practices website at <http://www.ibm.com/developerworks/db2/bestpractices/>.

Goals of good database storage design

Good database storage design has the following desirable characteristics:

- Predictable I/O and system performance
- Uniform use of available I/O bandwidth and capacities—avoiding “hot-spots”
- Ease of on-going management—for example, adding new storage
- Ease of problem determination
- High availability through redundancy

Simplicity in database storage design

**“Make everything as simple as possible, but not simpler”
– Albert Einstein**

In the myriad choices one can make when designing storage for a database, simplicity is the system architect’s and DBA’s secret weapon. The best practices presented in this document lay out straightforward rules of thumb that will assist in achieving all the goals of good database storage design.

This simplicity sometimes comes at the cost of not choosing the most optimal I/O characteristics for a specific table or table space. It is always possible that an experienced DBA with excellent storage skills and an unlimited amount of storage administrator’s time could carve out storage for a particularly important table or index from specific portions of physical disks. The problem with this approach is that, while it might provide the best performing solution at the time it is devised, in order to maintain the original design objectives, it will nearly always result in a system that is more complex to manage. Problem determination will nearly always be difficult—the storage bandwidth that was initially considered sufficient for the particularly important table or index may not be as sufficient as time goes on and the application grows.

The point of good database storage design is that, on a dynamic system, all goals should be met as part of the initial design of the system, and should continue to do so over the time that the database is in use. The simple best practices described in this document achieve these goals with virtually no performance sacrifices.

Recipe for database storage success

Think about real physical disks, not just storage space

Physical disks, which are connected to the storage controller, are not directly accessible by host systems, such as DB2 database servers, and are not directly visible by DBAs. Storage administrators provision units of storage as logical unit numbers¹ (LUNs), which appear to host systems as bonafide SCSI disks. A LUN, however, is a completely virtual entity, provisioned by a storage administrator, and can map to any combination of physical disks. An individual LUN might be a single RAID array, a portion of a RAID array, a single physical disk, a portion of a disk or a “meta” of multiple RAID arrays.

While the storage world might have become more virtualized, the reality is that data is still stored on mechanical disk drives. It does not matter which vendor’s storage subsystem is used; ultimately the data resides on mechanical disk drives, that is, on rotating physical disk platters. The bandwidth of storage that can be achieved from a provisioned LUN is directly proportional to the number of actual physical disks of which the LUN is comprised.

While storage controller caches assist with improving storage bandwidth, DB2 database systems already cache relevant data in their buffer pools. This makes it less likely that storage controller caches can sufficiently reduce the need for real physical disks in order to support I/O intensive systems, such as DB2 database servers. In an enterprise database system, which is often I/O intensive, the end result is that there is absolutely no substitute for actual physical disks.

In addition to traditional SAN storage controllers, additional layers of storage virtualization are being added in the enterprise that further abstracts the physical disk from the DBA. Examples of such virtualization include the San Volume Controller (SVC) and AIX® VIOS. These forms of virtualization can provide desirable function enhancements, such as the ability to migrate, transparently, from one set of LUNs to another, or the ability for multiple host LPARs to share a fiber channel Host Bus Adapter (HBA). They do so however at a cost that generally includes additional sub-systems in the I/O path. Additionally, they do not reduce the need for real physical disks for I/O intensive systems.

Dealing with highly virtualized storage

As was noted in the introduction of this paper, disk storage is more and more often treated as a generic commodity, where the storage space available is often abstracted from the physical devices on which it resides.

If your enterprise’s I/O infrastructure requires that such storage systems be used, DBAs need to continue to ensure that the virtual LUNs that are provided are indeed made up of dedicated physical disks. Here’s why: an enterprise system quickly becomes I/O

¹ In SCSI storage systems, a logical unit number (LUN) is a unique identifier used on a SCSI bus that enables it to address up to 8 separate devices (such as a disk drive), each of which is a logical unit.

bound if there are too few real disks to keep the CPUs busy. Unfortunately, while those of us concerned with database performance prefer to communicate our storage needs in terms of the number of real disks, storage administrators, like much of the rest of the world, often think of storage needs merely in terms of space. And with platter sizes having increased substantially over the last decade or so, the challenge of making an argument for more physical disks per CPU core rather than just more space has only become more difficult.

To make matters even more challenging, database administrators who gain access to the number of physical disks they need to ensure good performance might find themselves having to justify having *too much* space. For example, let's say you have a single CPU core with storage on 20 physical disks. This ratio of disks-to-CPU's should yield sufficient I/O parallelism to provide great performance. However, if each disk device can store 150 GB, you have approximately 3 TB worth of space per CPU core. If you have multiple CPU cores, each with a 1:20 ratio of CPUs to physical disks, you can see how the total amount of storage can quickly add up.

Despite having all of this "free" space, it is important that this storage not be over-committed. For example, you might be tempted to allocate some of the unused storage in such a deployment to other applications or processes. Keep in mind, though, that too many I/O-operations-per-second (IOPS) from competing applications or processes can cause performance for any one application to degrade. This means that storage administrators should resist the temptation to provide the unused space as a separate LUN for other applications over which you, the DBA have no control.

Now, you could quite reasonably use the unused space yourself as a staging area for online backups or archive logs for your database before they are backed-up to long-term storage. This can be a good use of the unused space because it is entirely within your control when backups are performed. In other words, it's entirely within your (as opposed to other, unknown users' or applications') control when these devices are used; you can perform online backups during periods when peak I/O throughput is not required.

If you use such a strategy to maximize space use, keep in mind there is inherent risk to using the same disks for data and for backups. Ensure that you archive your backups in a timely manner to an external backup target, for example, Tivoli® Storage Manager (TSM).

Since CPU speeds are expected to continue to increase (in the form of additional processing parallelism, i.e. additional CPU cores, rather than clock frequency), the expected trend is that more and more physical disks will be required per system to ensure that a database server does not become I/O bound. Thus, it is more important than ever before that DBAs ensure that they are eliminating as many I/O operations as possible through good schema design and by exploiting advanced functionality in DB2 database systems, such as MDCs, MQTs, and compression.

Given that CPU processing speeds have increased a substantial amount relative to spindle speeds, a good rule of thumb is to ensure that there are 15 to 20 dedicated

physical disks per CPU core. This number can potentially be reduced by using I/O techniques such as Multidimensional Clustering (MDC), Materialized Query Tables (MQT), and good schema management and design.

It is worth noting that at the time this paper was written, the number of physical disks stated here assume processor and disk technology that is common in the enterprise. This includes IBM POWER5™, Intel® Xeon® and AMD® Opteron™ processors. Common spindle speeds are 15000 rpm. As the next generation of processors becomes common, it is expected that a larger number of physical disks will be required per processor core for I/O intensive database servers.

Have dedicated LUNs and file systems per non-DPF DB2 database server / per DPF partition

It is best not to share LUNs and physical disks between DB2 servers/partitions. Indeed, it is a best practice to have dedicated LUNs for each non-DPF DB2 database server and for each DPF database partition.

Dedicating a LUN to a DB2 server or partition does preclude the physical disks that comprise that LUN from being used to create separate LUNs that could be used for purposes that are unlikely to interfere with the primary use of those disks. As mentioned in the previous section, though, you should ensure these LUNs are under your control and used carefully. The previously discussed example of using the excess space as a staging area for backups and archived logs could be one such purpose.

A single file system should be created on each such LUN and should be dedicated for use by a single DB2 server or DPF database partition.

Dedicated LUNs and dedicated file systems per LUN keep the storage layout simple and can assist with problem determination.

For DPF systems, it is recommended that the IBM Balanced Configuration Warehouse practices are followed.

An example of how this technique can help with problem determination is when poor selection of a partitioning key on a table prevents a query from getting the right amount of parallelism. When LUNs and file systems are dedicated to database partitions, this problem becomes evident when you see that one set of LUNs is busy for much longer than the other LUNs, because one partition has all of the data that needs to be processed, while other partitions have relatively little data.

Stripe at most in two places

Storage controllers offer excellent RAID striping directly in the controller firmware. Enterprise systems should be designed to use the striping provided by storage controllers. A convenient way to do this is to have the storage controller expose an individual RAID array, for example, RAID-5 or RAID-10, as a single LUN. One or more of such LUNs can then be provided to DB2 partitions.

When more than one LUN is provided to a DB2 database server or DPF database partition, use the DB2 database system's fine-grained striping between containers.

Because two levels of striping are adequate for all systems, avoid using a third level of striping, such as the operating system's logical volume manager. Logical volume manager (LVM) striping is beneficial to other middleware, which, unlike DB2 database systems, lack the capability to do its own striping.

Separate DB2 transaction logs and data

To help achieve best availability, separate the transaction logs and DB2 data, or table spaces, onto separate physical disks and onto separate LUNs.

Each DB2 partition should be provided one LUN with dedicated physical disks for its transaction logs and, typically, multiple LUNs for its table space containers or data.

While the ratio of log physical disks to data physical disks is heavily dependent on workload, a good rule of thumb is 15% to 25% of the physical disks for logs, and 75% to 85% of the physical disks for data.

Use file systems instead of raw devices— one file system per LUN

Direct and concurrent I/O have almost completely eliminated the need to use raw logical volumes for performance. File systems provide superior manageability to raw devices, as a single file system can be used as a container for multiple table spaces.

Each LUN that is provisioned for a database partition should be used to create a separate file system for use by that partition, that is, one file system per LUN.

Each DB2 partition will then generally have:

- One transaction log file system—created using the LUN that was provisioned for that partition's transaction logs.
- More than one data file system—each file system created using its own separate LUNs that were provided for table space data.

Where possible, use RAID-10 for transaction logs, RAID-10 or RAID-5 for data

RAID-10 provides excellent redundancy because it can survive multiple physical disk failures. RAID-5, on the other hand, can only survive a single physical disk failure. However, the tradeoff for the increased redundancy of RAID-10 is its significantly higher cost compared to that of RAID-5.

You will achieve greater resiliency to disk failures if you use RAID-10 for both logs and data. If you can only use RAID-10 for storing *either* data or logs, use it for logs, and use RAID-5 for data. If you choose to use RAID-5 for both logs and data, you still have a very resilient storage configuration; however, you may find you have a somewhat greater dependency on your database backups. Set the table space `EXTENTSIZE` to the RAID stripe size (or failing that, to a size conducive to big-block reads)

The amount of contiguous data on each physical disk in a RAID array is known as a "strip" and the amount of data across the array that comprises all the strips in a single array is called a "stripe".

Typical RAID strip sizes are 32kb, 64kb, 128kb, and so forth.

A RAID-5 4+1 array would have a stripe size equal to 4 times the strip size.

The `EXTENTSIZE` for table spaces should be set to a number of pages that includes an entire RAID stripe.

For example, on a system with a 128kb strip size that uses RAID-5 4+1, the stripe size will be 512kb (128kb x 4). If the page size used is 8kb, then an `EXTENTSIZE` of 64 (512kb/8kb) is appropriate.

Setting the table space extent size in virtualized storage environments

Sometimes it is not possible to know which RAID array the file system for a given DB2 table space container is stored on. This can happen if there are layers or storage virtualization between the LUN that is provisioned to a database server host machine and the storage controller. In such cases, when you create your table spaces, the `EXTENTSIZE` should be set to a number that is conducive to efficient big-block I/O performed by prefetchers. A good rule of thumb is having an extent size that is 256 KB. Extent sizes of 128 KB or 512 KB are also good options.

The default settings for `EXTENTSIZE` (32 pages, as specified by the `DFT_EXTENT_SZ` configuration parameter) should provide adequate performance in most cases. For example, if 8 KB pages are used for the database, an `EXTENTSIZE` of 32 (8 KB x 32 pages = 256 KB) should provide an adequate setting for `EXTENTSIZE` when details regarding RAID strip sizes are not known. Even with a 4 KB or 16 KB page size, an `EXTENTSIZE` of 32 pages still yields an extent size of 128 KB or 512 KB, respectively, which is within the recommended range.

Set DB2_PARALLEL_IO for optimal I/O parallelism

By default, the I/O servers, or *prefetchers* for DB2 for Linux, UNIX and Windows perform one extent-sized I/O for each table space container during table scans. `EXTENTSIZE` is thus not only DB2's unit of striping; it is also the read I/O size that prefetchers use during sequential prefetch

If `EXTENTSIZE` is set as per the best practice in the previous section, to ensure that one extent of data spans all the drives within the (single) RAID array that comprises the file system that each DB2 container uses, then `DB2_PARALLEL_IO` does not need to be set, because the database manager will automatically prefetch the extent from all physical disks in a container in parallel.

There are other ways to set `EXTENTSIZE` and to design the striping of a DB2 system than what is presented in this paper, however.

One alternative used in some configurations is to set `EXTENTSIZE` to span continuous data on a single physical disk within each RAID array. That is, to set `EXTENTSIZE` to be the *strip* size, rather than the stripe size as recommended in the previous section. In such

configurations, a single extent-sized I/O per table space container during sequential prefetch is inadequate as it will drive only a single physical disk within the RAID array that the file system is based on. Setting `DB2_PARALLEL_IO` is required for such systems if you want the database manager to generate multiple extent-sized prefetch requests to drive all of the physical disks for each DB2 table space container in parallel.

`DB2_PARALLEL_IO` allows the user to explicitly specify the number of physical disks under each container so as to generate the right number of prefetch request for each container. For example, if each table space container exists on a file system that is backed by a RAID 5 4+1 array, the following is the appropriate `DB2_PARALLEL_IO` setting:

```
db2set DB2_PARALLEL_IO=*,5
```

The “*” indicates that the setting applies to all table spaces. The 5 indicates that under each container are 5 (4+1) individual physical disks that should each be driven using an extent sized read request by a separate prefetcher.

The `DB2_PARALLEL_IO` setting is taken into account by the algorithms that compute:

- The prefetch size for each table space, when the `PREFETCHSIZE` option on the `CREATE` or `ALTER TABLESPACE` statement is set to `AUTOMATIC`
- The number of I/O servers, or prefetchers, when the `num_ioservers` configuration parameter is set to `AUTOMATIC`.

(See “Do not hand-tune the `NUM_IOCLEANERS`, `NUM_IOSERVERS`, and `PREFETCHSIZE` configuration parameters” on page 14 for a related recommendation.)

*Use the **NO FILE SYSTEM CACHING** clause*

The `NO FILE SYSTEM CACHING` clause enables direct or concurrent I/O, whichever is applicable to the operating system platform on which the DB2 database system runs. Direct or concurrent I/O effectively allows DB2 I/O operations to have raw device-like performance on file systems.

Support for the `NO FILE SYSTEM CACHING` clause was added to the `CREATE TABLESPACE` and `ALTER TABLESPACE` statements in DB2 Universal Database, Version 8.2. Since Version 9.5, it has been the default for newly created databases for those file systems where direct or concurrent I/O is available, such as JFS2, GPFS™, and VxFS.

*Use **DB2 automatic storage to stripe everything everywhere***

DB2 automatic storage (AS) technology is a simple and effective way to provision storage for a database. Storage is provided to the database, rather than to individual table spaces, using the `CREATE DATABASE` command. For example:

```
DB2 CREATE DATABASE MYDB ON /data1, /data2, /data3 DBPATH ON
/mydbpath
```

This example command creates a database with three storage paths: `data1`, `data2`, and `data3`. Each of these three paths is a separate file system, each created using dedicated LUNs. (Note: Unless you specify otherwise, automatic storage is used by default when you create a database using the `CREATE DATABASE` command.)

The `DBPATH` parameter on the `CREATE DATABASE` command is set to another separate file system (`/mydbpath`) created using the LUN provisioned for the DB2 transaction logs. Transaction logs and DB2 metadata all reside on this file system.

The default table spaces created by the DB2 database system (`SYSCATSPACE`, `USERSPACE1` and `TEMPSPACE1`) each have three containers—one on each of the three storage paths.

Any new table spaces created without containers explicitly provided are also created using a stripe everything-everywhere approach.

For example, consider the following DB2 statement:

```
DB2 CREATE TABLESPACE MYTS
```

The table space `MYTS`, created using the `CREATE TABLESPACE` statement, will also have three containers, one on each of the storage paths.

While using automatic storage does not preclude you from defining system managed space (SMS) or database managed space (DMS) table spaces, or files, in the same database, automatic storage generally eliminates the need for you to define them.

Because the DB2 storage manager uses a simple stripe everything-everywhere approach, the best practice with automatic storage is to use storage paths, or file systems, that are uniform in capacity. This ensures that parallelism remains uniform as one storage path does not fill up prematurely thus causing some parts of some table spaces to have a different striping width from others.

When space needs to be added to a database, try to extend all existing paths uniformly. That is, increase the capacity of each file system by the same amount.

If you cannot extend the storage paths uniformly, add a new set of (uniform) storage paths using the `ADD STORAGE ON` clause of the `ALTER DATABASE` command. The new set of storage paths should have the same I/O capabilities as the original set. Ideally the same number of storage paths that were defined before should be added together.

For example, to add space to the database `MYDB` that was created earlier, the best option is to increase the capacity of the file systems `/data1`, `/data2`, and `/data3` by the same amount.

If that is not possible, then a new set of storage paths (file systems), that have the same I/O characteristics as the original set should be added:

```
DB2 ALTER DATABASE ADD STORAGE ON /data4, /data5, /data6
```

Because the original storage paths are uniform in size, when they fill up, they all fill up together and table spaces that need additional storage get a new stripe set of containers—one for each of the new storage paths.

Do not hand-tune the NUM_IOCLEANERS, NUM_IOSERVERS, and PREFETCHSIZE configuration parameters

The default value for these parameters is `AUTOMATIC`. The DB2 database manager does an excellent job in selecting appropriate values for these parameters; therefore, they generally need not be hand-tuned.



Best Practices

- Use 15-20 DEDICATED physical disks per CPU core
- Ensure you have control of the use of unused space for your LUNs
- Dedicate LUNs to non-DPF DB2 database servers / DPF database partitions
- Stripe at most in two places
- Separate DB2 transaction logs and data on to separate physical disks and LUNs
- Use file systems instead of raw devices—create one file system per LUN
- Use RAID-10 for transaction logs, RAID-10 or RAID-5 for data LUNS
- Set your table space `EXTENTSIZE` to RAID stripe size; if your `EXTENTSIZE` is the same as the *strip* size, consider using the `DB2_PARALLEL_IO` configuration parameter to increase prefetch parallelism.
- Use direct or concurrent I/O by using the `NO FILE SYSTEM CACHING` clause
- Use DB2 automatic storage for everything-everywhere striping
- Use `AUTOMATIC` (the default) for `NUM_IOCLEANERS`, `NUM_IOSERVERS` and `PREFETCHSIZE`

Conclusion

Manually mapping individual database tables and indexes to individual physical disks and parts of physical disks is an exercise best left to decades past. The key to taming the complex, virtual, networked storage beast is keeping database storage design as simple as possible.

While it seems counterintuitive at first, complexity is always best tackled with simplicity rather than with more complexity. While simplicity is not always easy, the best practices provided in this document provide an easy-to-follow recipe for database storage design success.

Above all, a DBA's time and effort is best spent optimizing the database schema, rather than physical database storage. Optimizing the schema involves using functionality like MDCs, MQTs, creating appropriate indexes, and dropping inappropriate ones. After all, there is no higher throughput or lower latency I/O than one that does not need to be performed at all.

Further reading

- DB2 Best Practices
<http://www.ibm.com/developerworks/db2/bestpractices/>
- IBM DB2 9.7 for Linux, UNIX, and Windows Information Center
<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp>
- DB2 9 for Linux, UNIX and Windows manuals
<http://www.ibm.com/support/docview.wss?rs=71&uid=swg27009552>
- DB2 Data Warehouse Edition documentation
<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.dwe.welcome.doc/dwe91docs.html>
- IBM Data Warehousing and Business Intelligence
<http://www.ibm.com/software/data/db2bi/>

Contributors

Danny F. Arnold

DB2 Technical Evangelist

John Bell

Chief Data Warehousing Solution Architect

Whit Smith

Certified IT Specialist

Linda Snow

Executive IT Specialist

Reuven Stepansky

Senior Managing Specialist

North American Lab Services

Tim Vincent

Chief Architect DB2 LUW

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

Without limiting the above disclaimers, IBM provides no representations or warranties regarding the accuracy, reliability or serviceability of any information or recommendations provided in this publication, or with respect to any results that may be obtained by the use of the information or observance of any recommendations provided herein. The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any recommendations or techniques herein is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Anyone attempting to adapt these techniques to their own environment do so at their own risk.

This document and the information contained herein may be used solely in connection with the IBM products discussed in this document.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual

results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Intel Xeon is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.