IBM DB2 for Linux, UNIX, and Windows

Best Practices Physical Database Design for Online Transaction Processing (OLTP) environments

Authors

The Physical Database Design for Online Transaction Processing (OLTP) environments white paper was developed by the following authors:

Vincent Kulandai Samy

DB2® HADR Development
Information Management Software

Weilin Lu

DB2 for Linux, UNIX, and Windows QA
Information Management Software

Amyris Rada

Senior Information developer
DB2 Information Development
Information Management Software

Punit B. Shah

Senior Software Engineer
DB2 Next Generation Data Analytics

Sripriya Srinivasan

Advanced DB2 Technical Support Engineer
Information Management Software

Contents

Figures

Executive Summary

Understanding the basic concepts, the stages of physical database design, and the advanced aspects that affect the structure of databases is key for a successful database design.

This paper focuses on physical database attributes that are affected by the specifics of DB2 database servers in online transaction processing (OLTP) environments.

Introduction to physical database design

The main objective of physical database design is to map logical database design to the specific features and functions of the actual database server, in this case a DB2 database server.

Database design consists of the following three phases:

1. Designing a logical database model. This phase includes gathering of business requirements, and entity relationship modeling.
2. Converting the logical design into database objects. This phase includes table definitions, normalization, primary key (PK) and foreign key (FK) relationships, and basic indexing. It is often performed by an application developer.
3. Adjusting the deployed physical database design. This phase includes improving performance, reducing I/O, and streamlining administration tasks. It is often performed by a database administrator.

Following logical database design, physical database design covers those aspects of database design that impact the actual structure of the database on disk. These aspects are described in phase 2 and 3. Although you can perform logical design independently of the relational database chosen, many physical database attributes depend on the target database server. Physical database design includes the following aspects:

- Data type selection
- Table normalization
- Table denormalization
- Indexing
- Clustering
- Database partitioning
- Range partitioning
- Memory provisioning
- Database storage topology
- Database storage object allocation

For details about *Database storage topology* and *Database storage object allocation*, see "DB2 Best Practices: Database Storage" at *http://www.ibm.com/developerworks/data/bestpractices/databasestorage/*.

Designing a physical database model is a process that requires a periodic review even after the initial design is rolled out into a production environment. New and emerging business methodology, processes, and change requirements affect an existing database design at architectural level. The best practices for database physical design described in this paper are relevant to both new deployments and existing deployments.

Today, we can achieve I/O reductions by properly partitioning data, distributing data, and improving the indexing of data. All of these innovations that improve database capabilities expand the scope of physical database design and increase the number of design choices resulted in the increased complexity of optimizing database structures. Although the 1980s and 1990s were dominated by the

introduction of new physical database design capabilities, the subsequent years have been dominated by efforts to simplify the process through automation and best practices.

The best practices presented in this document have been developed for today's database systems and address the features and functionality available in DB2 Version 9.7 software.

OLTP workload characteristics

An important aspect of a physical database design is to identify characteristics of a workload because that determines overall direction for physical database design. This paper discusses only OLTP workload characteristics because the focus is on physical database design for OLTP environments.

DB2 has a number of features designed to meet any workload demand. Identifying the type of workload helps you in selecting the adequate DB2 features.

Some database applications are transaction-oriented. For example, buying an airline ticket or checking a flight status are transactions. Each transaction has certain response time requirement. For example, 10 milliseconds to check for a flight status, or 20 milliseconds to purchase a ticket. Several concurrent transactions can be active at any time; for example, an online retailer that is processing thousands of orders every second or servicing several concurrent online catalog browsing queries.

From query engine perspective, such workload translates into smaller queries which are measured by amount of data movement. OLTP workloads have a mix of readers such as SELECT SQL statements and writers such as INSERT, UPDATE, and DELETE (IUD) SQL statements executed by several active applications. In addition, an OLTP workload has the following typical characteristics:

- Concurrent applications that touch a disjoint set of data. This action results into random I/O and stress the I/O subsystem.
- No significant amount of serial I/O operations.
- Heavy demand on transaction logging devices by IUD statements can become a bottleneck.
- Stricter application of isolation levels puts a higher demand on locking infrastructure.
- Disjoint sets of data and the seldom reuse of data leads to a large working set size that results in a low buffer pool hit ratio and frequent page cleaning.
- Relatively simple queries that do not include complex joins or the ORDER BY clause.
- Stringent uptime requirements. For example, database systems must be available 24x7.

There are additional workload classifications such as data warehousing. The workload characteristics in data warehousing include mostly read-only operations (SELECT statements), long running queries that access a large amount of data, and queries that involve complex multitable joins, data aggregation, and weaker isolation requirements.

In many instances, there is no clear boundary that distinguishes one workload kind from another one. A substantial number of workloads exhibit mixed characteristics. For example, a blend of OLTP and data warehousing workload characteristics. In such cases, mixing physical database design principles from both types of workloads is the best approach.

Physical database design

A high-quality physical database design is an important factor in a successful database deployment.

The choices and decisions made during physical database design have a long lasting effect and far reaching impact in terms of overall database health and day-to-day administrative overhead incurred in a data center. Understanding DB2 features and how it can be applied to meet the business needs is crucial to come up with a high-quality physical database design that can adapt to evolving business requirements over time.

A high-quality physical database design must consider the following items:

- Business service level agreement (SLA)
- I/O bandwidth
- Performance objectives such as response time and throughput
- Recovery time
- Maintenance window
- Administrative overhead
- Reliability, availability, and serviceability (RAS)
- Data (lifecycle) management

As your business requirements change, you must reassess your physical database design. This reassessment should include periodic revisions of the design. If necessary, make configuration and data layout changes to meet your business requirements. For example, if the recovery point objective (RPO) and recovery time objective (RTO) parameters change with respect to the original design, consider using the DB2 High Availability and Disaster Recovery (HADR) feature or spread tables across more table spaces so that table space restore and roll forward operations can be performed in shorter amount of time.

A high-quality physical database design tries to achieve the following goals:

- Minimize I/O traffic.
- Balance design features that optimize query performance concurrently with transaction performance and maintenance operations.
- Improve the performance of administration tasks such as index creation or backup and recovery processing.
- Reduce the amount of time database administrators spend in regular maintenance tasks.
- Minimize backup and recovery elapsed time.
- Reassess overall database design as business requirements change.

Data modeling

Gathering requirements and creating a logical model are the key to a good physical database design.

The first step for data modeling is gathering requirements. This step involves identifying critical business artifacts, data, and information that requires maintenance. Such business artifacts are called entities. For an online shopping catalog, information about customers, products, and pricing are examples of business critical information or entities.

The requirements are gathered by stakeholder input. The requirements and data model are further refined along the way feeding into each other in iterative manner to create a logical model. The Figure 1 shows the iterative data modeling paradigm:

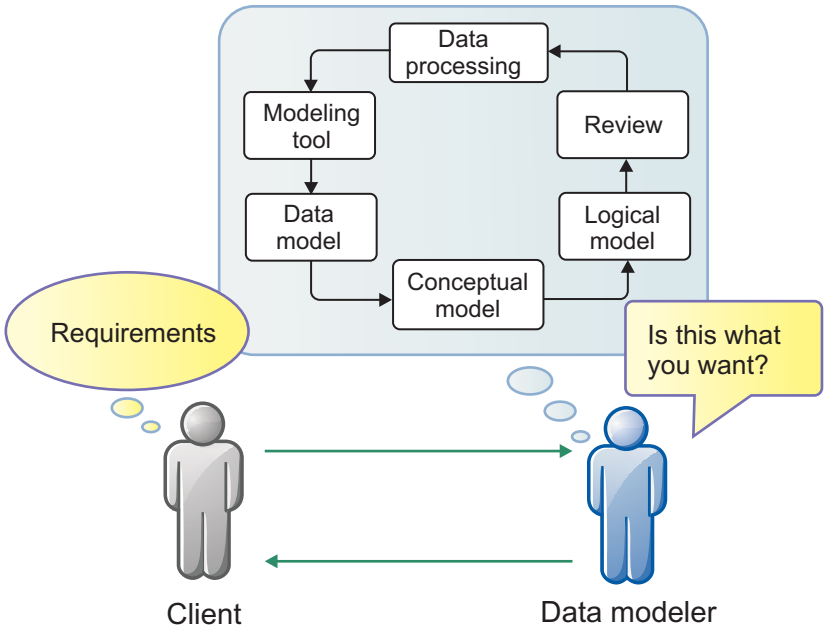

Figure 1. Logical data model

After gathering requirements, further structuring and organizing the data is required. Data modeling can define and organize the data, and can impose, implicitly or explicitly, constraints or limitations on the data placed within a structure. For example, an account holder in a bank customer management system must be associated with at least one account. No more than six withdrawals per month are allowed from savings accounts. Such conditions are constraints that are eventually reflected as a referential integrity constraints or other types of constraint in a relational database.

IBM InfoSphere Data Architect

IBM® InfoSphere® Data Architect is a collaborative data design tool that helps you discover, model, relate, and standardize diverse and distributed data sources. You can use InfoSphere Data Architect to create a data model. This model can eventually be used to create databases and database objects, including tables, indexes, and table spaces.

The data model design lifecycle helps you conceptualize and develop your data model, by using an iterative design process.

The forward-engineering method builds data models and databases from scratch, whereas the reverse-engineering method uses existing data models and databases to create models. You can use InfoSphere Data Architect to reverse-engineer physical data models from existing databases and schemas. Also, you can use the transformation tool to create logical data models to further refine your projects.

Logical to Physical database design

You can use InfoSphere Data Architect to create a logical data model and then transform it into a physical data model.

Physical data models are logical data models that are specific to a database, and they are based on database specifications. InfoSphere Data Architect supports physical data modeling for databases in DB2 for Linux, UNIX, and Windows software, DB2 for z/OS® software, and DB2 for i software.

With a physical data model specific to a DB2 database product, you can model storage that includes column data types, partitions, table spaces, indexes, or buffer pools, in addition to other storage objects. For more details about data modeling, see "Database Fundamentals" at *http://www.ibm.com/developerworks/wikis/display/db2oncampus/FREE+ebook+-+Database+fundamentals* and "Getting started with IBM InfoSphere Data Architect" at *http://public.dhe.ibm.com/software/dw/db2/express-c/wiki/Getting_Started_with_IDA.pdf*.

Best practices

Use the following design best practices for data modeling:

Use InfoSphere Data Architect to perform data modeling and database physical design tasks such as:

- Create a logical data model and then transform it into a physical data model. Work with the physical data model to plan the physical storage for table spaces, indexes, or views by adding storage objects.
- Generate DDL scripts that will help you to deploy the DB2 database. Run these script to create the database and its objects on DB2 server.
- Revise your physical data model as your business needs change and make changes to the data model accordingly.

Storage systems

Storage systems offer many advantages over individual disks, including reducing storage administrative overhead, better performance, huge storage server cache, multipathed access, battery backup, improved reliability, and improved availability.

Instead of individual disks, it is common these days to have mid-range to high-range storage systems such as IBM System Storage® DS6800 and DS8300 servicing a DB2 database server.

Despite recent success of solid-state devices (SSD), magnetic disks are still the norm in data centers. Because of the gap between processor speed and disk bandwidth, disk I/O bandwidth quickly becomes a bottleneck in high performance database systems deployments. One of your planning goals should be that the database engine is not I/O bound. To keep the database engine from being I/O bound, minimize the I/O that a DB2 server performs and distribute the data over several disk spindles.

Disk arrays

Disk arrays are rated at a relatively higher mean time to failure (MTTF) and mean time between failures (MTBF) than disk drives, primarily due to the testing methods and the type of statistical models used to rate the disk drives. Nonetheless, the reality is that every now and then, a disk drive fails in a database system. Therefore, you should have some level of redundancy for disk drives. Without the redundancy, disk drive failure would be disastrous requiring restore and roll forward to recover a lost table space.

Most storage systems and operating systems support the redundant array of independent disks (RAID) feature. There are several RAID levels, each RAID level indicates how disk arrays are arranged and what faults are tolerated, which indirectly influences aggregate performance of disk array.

RAID0 uses disk blocks in round-robin fashion, also called "striping". There is no redundancy for RAID0, however it speeds up write and read operations because all disks that belong to a disk array can be accessed in parallel during I/O operations.

RAID1, also called disk mirroring, requires a redundant disk for each disk in a disk array. RAID1 provides best fault tolerance. Half of the mirrored disks in an array can be lost without any effect to the database system. However, using RAID1 carries a cost because it doubles the disk space requirements and lowers write throughput because every write operation needs to be performed twice.

RAID2 and RAID3 use a bit-level and a byte-level parity approach for data redundancy. Both RAID4 and RAID5 use parity block for data redundancy, thus offering fault tolerance from one disk drive failure. RAID4 uses a dedicated parity disk, during intensive write operation which is typical of OLTP workloads, this parity disk can become severely bottle-necked. RAID5 offers an improvement by using distributed parity to eliminate the bottleneck during write operations.

For OLTP workloads, use RAID5 for DB2 table space containers. If possible, use storage-server-level or adapter-level hardware RAID. For disk arrays, disable

storage server level read-ahead because an OLTP workload does not exhibit a sequential pattern. Enable write-behind for quicker write turnaround. Write-behind does not wait to write a page to disk. As soon as a page is copied into the storage sever cache, the write operation is considered successful from the DB2 database manager point of view. Write-behind is not a problem in case of power failure. A battery backup allows the storage server to flush pages in the storage cache to the disks.

Each change in a database is recorded in log files as a log record. Each log record has information to redo or undo the change in the database. This log is an important feature in databases to maintain data integrity and the atomicity, consistency, isolation, and durability (ACID) properties.

OLTP workloads involve insert, update, and delete operations that put heavy demand on logging I/O performance. Reliability, availability, and scalability (RAS) and performance are the two most important requirements for logging. Although RAID5 can be used for the DB2 transaction logging active log path, due to critical nature of transaction logging, higher level of redundancy such as RAID1 should be used. If RAID1 performance is an issue, use RAID 0+1 for logging devices because this RAID level provides disk mirroring and striping. Striping distributes the data among the disks in the array.

Another feature offered by modern storage systems and operating systems is load balancing and failover capabilities for host channel adapters. Host channel adapters, also called multipath I/O, connect servers and operating systems to storage. During normal runtime, adapters share a workload, and if an adapter becomes inoperative, another adapter continues servicing the database I/O, with little or no impact on performance.

Best practices

Use the following design best practices for storage systems:

- Use storage server level hardware that has RAID array capability. RAID5 offers a balance between cost, redundancy, and performance.
- Disable storage-server level read-ahead since OLTP workloads do not exhibit sequential I/O and do not benefit from read-ahead.
- Use RAID 1+0 or RAID5 as log devices for better performance and higher RAS.
- If the storage system has a battery backup, enable write-behind.
- If the hardware level RAID support is not available, use logical volume manager level RAID.
- Use as many hardware level RAS features and performance capabilities as possible. For example, hardware RAID features tend to be faster than software RAID features in the operating system or volume manager level.

For more details about storage systems best practices, see “Best Practices: Database storage” at *http://www.ibm.com/developerworks/db2/bestpractices/databasestorage/*.

Table spaces and Buffer pools

When designing table spaces and container placement on physical devices, the goal is to maximize I/O parallelism, increase buffer utilization, and increase buffer pool hit ratio. To achieve that goal, you need a thorough understanding of the database design and applications.

Understanding how table spaces and buffer pools work and influence overall performance of a database helps you determine such issues as whether segregating two tables to different devices leads to parallel I/O, or whether a table should be created in a separate table space so it can be fully buffered.

The two main storage areas to consider in your design are:

1. Table spaces. The type and design of your table space determines the efficiency of the I/O performed against that table space.
2. Buffer pools. Most page data manipulation takes place in buffer pools, configuring buffer pools is the single most important tuning area.

Table space design for OLTP workloads

The type of workload that the database manager manages in your environment significantly influences choice of what type of table space to use and what page size to specify. DB2 databases support variety of page sizes for a table space such as 4 KB, 8 KB, 16 KB, and 32 KB.

There are three types of table spaces you can choose from for DB2 databases:

- Managed by automatic storage.
- Managed by the database. Also called database managed space (DMS).
- Managed by the system. Also called system managed space (SMS).

For details about tables spaces, see "Table spaces" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.dbobj.doc/doc/c0004935.html*.

OLTP workloads are characterized by transactions that need random access to data. OLTP transactions often involve frequent insert or update activity and queries which typically return small sets of data. When table space access is random and involves one or a few pages, prefetching is less likely to occur. DMS table spaces using device containers perform best in this situation. DMS table spaces with file containers, or SMS table spaces, are also reasonable choices for OLTP workloads if maximum performance is not required. Using DMS table spaces with file containers, where FILE SYSTEM CACHING is turned off, can perform at a level comparable to DMS raw table space containers.

When a table space has FILE SYSTEM CACHING turned off, the database manager chooses between concurrent I/O (CIO) and direct I/O (DIO), in that order, depending on the underlying file system support. Most operating systems and file systems support DIO or CIO. CIO is improved version of DIO and offers better performance than DIO. Like raw devices, CIO or DIO file system containers bypass the file system buffer. But unlike raw devices, they are easier to manage. For more details about using CIO or DIO in table spaces, see "New table space

containers use concurrent I/O or direct I/O by default" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.dbobj.doc/doc/c0052534.html*.

Starting with DB2 Version 9.5, the NO FILE SYSTEM CACHING setting is the default for new databases for those file systems where DIO/CIO is available. If you are migrating from a Version 9.1 database, consider the impact of this change.

With little or no sequential I/O expected, the settings for the EXTENTSIZE and the PREFETCHSIZE parameters on the CREATE table space statement do not have a substantial effect on I/O efficiency. The value of the PREFETCHSIZE parameter on the CREATE table space statement should be set to the value of the EXTENTSIZE parameter multiplied by the number of device containers. Alternatively, you can specify a prefetch size of -1 and the database manager automatically chooses an appropriate prefetch size. This setting allows the database manager to prefetch from all containers in parallel. If the number of containers changes or there is a need to make prefetching more or less aggressive, change the PREFETCHSIZE value accordingly by using the ALTER table space statement.

Table space page sizes

For OLTP applications that perform random row read and write operations, use a smaller page size because it does not waste buffer pool space with unwanted rows. However, consider the following important aspects of the page size selection.

- **Row size greater than page size.** In this case, you must use a larger page size. When considering the size of temporary table spaces, remember that some SQL operations such as joins can return a result row that does not fit in the table space page size. Therefore, you should have at least one temporary table space with a 32 KB page size.
- **Higher density on disk by choosing a larger page size.** For example, only one 2100 byte row can be stored in a table space with 4 KB page size, which wastes almost half of the space. However, storing the row in a table space with 32 KB page size can significantly reduce this waste. The downside of this approach is the potential of incurring in higher buffer pool storage costs or higher I/O costs. Choose the largest page size with a storage cost that you can afford.

Data placement in table spaces

The following recommendations are general advice for table space data placement:

- Create database objects that need to be recovered together in the same table space for easier backup and restore capabilities. If you have a set of database objects such as tables and indexes that are queried frequently, you can assign the table space in which they reside to a buffer pool with a single CREATE or ALTER TABLESPACE statement.
- Assign a buffer pool to temporary table spaces for their exclusive use to increase the performance of activities such as sorts or joins. Create one system temporary table space for each page size. The DB2 database manager chooses the temporary table space by using an internal algorithm based on the buffer pool size. Use SMS table spaces for temporary table spaces.
- Define smaller buffer pools for seldom-accessed data or for applications that require random access into a large table. In such cases, data does not need to be kept in the buffer pool for longer than a single query.
- Store LOB or LONG data in SMS or DMS file containers so that file system caching might provide buffering and, as a result, better performance. In general,

the database manager does not cache large data in buffer pools and must retrieve it from disk for applications that access either LOB or LONG data.

- Use FILE SYSTEM CACHING for the SYSCATSPACE table space to substantially benefit from file system caching because the system catalogs contain some LOB columns.
- For high-activity tables with LOB columns that are stored together with the data in table spaces with FILE SYSTEM CACHING, re-create these tables with the LOB columns stored in a separate table space which is using the file system cache for the I/O to avoid the possibility that database performance might be impacted due to demoted I/O. Another alternative is using inline data for these tables. For more details, see "Storing LOBs inline in table rows" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.dbobj.doc/doc/c0054525.html*.
- Create a single file system on each disk logical unit (LUN) and dedicate it to a single partition DB2 database. Dedicated LUNs and file systems per LUN keep the storage layout simple and can assist with problem determination. This is a customary best practice for single-partition DB2 databases.
- If you have many small tables in a DMS table space, you might have a relatively large amount of space allocated to store a relatively small amount of data. In such a case, you should specify a small extent size. If you have a large table that has a high growth rate, and you are using a DMS table space with a small extent size, you might have unnecessary overhead related to the frequent allocation of additional extents. Set the EXTENTSIZE parameter to the RAID stripe size.
- Use AUTOMATIC for the NUM_IOCLEANERS, NUM_IOSERVERS and PREFETCHSIZE 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 do not need to be hand-tuned.
- Using larger record identifiers (RID) increases the row size of your result sets for queries or positioned updates. If the row size in your result sets is close to the maximum row length limit for your existing system temporary table spaces, you might need to create a system temporary table space with a larger page size.

Buffer pool design

When designing buffer pools, you must understand the relationship between table spaces and buffer pools. IBMDEFAULTBP is the default buffer pool. The database manager also defines the IBMSYSTEMBP4K, IBMSYSTEMBP8K, IBMSYSTEMBP16K, and IBMSYSTEMBP32K system buffer pools, formerly known as the "hidden buffer pools".

Each table space is associated with a specific buffer pool. You should explicitly associate buffer pools to table spaces. If you do not associate a buffer pool to a table space, the database manager chooses the default buffer pool or one of the system buffer pools.

You should consider the following general guidelines for designing buffer pools in OLTP environments:

- Use the AUTOCONFIGURE command to obtain a good initial recommendation for buffer pool configuration.
- Use the self-tuning memory manager (STMM) and other automatic features to provide stability and strong performance. Enable STMM for automatic tuning of buffer pools in single partitioned environments.

Use STMM with caution in partitioned database environments because STMM does not make good recommendations for environments with skewed data distribution or for the tuning partition.

- Explicitly set the size of buffer pools that have an entry in the SYSCAT.BUFFERPOOLS catalog view.
- Associate different buffer pools for temporary table space and data table space to avoid possible buffer pool contention when temporary objects are accessed.
- Consider associating a separate buffer pool for large object data. Although, LOB data does not use buffer pools, the LOB allocator does use buffer pools.
- Choose your buffer pool page size based on table space page size. You cannot alter the page size after you create a buffer pool.

Buffer pool hit ratios are a fundamental metric for buffer pool monitoring. They give an important overall measure of how effectively the system is in using memory to reduce disk I/O. Hit ratios of 80-85% or higher for data and 90-95% or higher for indexes are typically considered good for an OLTP environment. These ratios can be calculated for individual buffer pools using data from the buffer pool snapshot or the **db2pd -bufferpools** command.

Keep frequently used read-only or read-mostly data in a single table space. Do not mix read-only or read-mostly with heavily write intensive (IUD) tables. It reduces a cache pollution by write intensive tables, which minimizes the chances of read-only or read-mostly pages being victimized when freeing space in a buffer pool. To free space in a buffer pool, unneeded pages are flushed to disk.

For a 64-bit system, the buffer pool can be almost any size. However, for most e-commerce OLTP applications that use a large database, tune the buffer pool size based on the buffer pool hit ratio. Bigger is still better, but at some point you experience diminishing returns as the buffer pool hit ratio moves to the over 98% range.

Page cleaning activity

Ordinarily, page cleaning drives a steady stream of page writes out to the table space containers in order to ensure the availability of buffer pool pages by subsequent table space reads. If the page cleaning is not effective, the agent itself can end up doing much of the cleaning. This often results in sporadic periods of intense write activity (bursty cleaning), possibly creating a disk bottleneck, alternating with periods of better I/O performance.

Use the following database configuration parameters and registry variables to tune page cleaning activity:

- Use the **num_iocleaners** configuration parameter to specify the number of asynchronous page cleaners for a database. Environments with high update transaction rates and large buffer pools might require more page cleaners to be configured. Set it to the number of physical storage devices used for the database. If the applications for a database consist primarily of transactions that update data, an increase in the number of cleaners speeds up performance. Increasing the page cleaners also decreases recovery time from soft failures, such as power outages, because the contents of the database on disk are more up-to-date at any given time. In DB2 Version 9.5 or later, extra cleaners beyond the recommended number can have a negative effect on performance.
- Use the **chngpgs_thresh** configuration parameter as the preferred way to affect the number of clean pages in the buffer pool. The **chngpgs_thresh** configuration

parameter specifies the percentage of changed pages at which the asynchronous page cleaners are started if they are not currently active. For databases with a heavy update transaction workload, you can generally ensure that there are enough clean pages in the buffer pool by setting the parameter value to be equal to or less than the default value. A percentage larger than the default can help performance if your database has few large tables. The default value of 60% is normally too high for OLTP workloads. A value between 20% and 40% is more appropriate. For example, if you had a 2 GB buffer pool, when 60% changed pages is reached, 1.2 GB (60% of 2 GB) would be written to disk as page cleaners are triggered. Writing this much data can cause an overall slow down in your system as the disk write happens. By setting the **chngpgs_thresh** parameter to a lower amount like 20%, the page cleaners are triggered more often, but less data is written to disk, and the slowdown might be unnoticeable by your users. Setting this parameter too low can result in excessive disk writes.

- Use the improved proactive page cleaning algorithm by setting the **DB2_USE_ALTERNATE_PAGE_CLEANING** registry variable to YES. This new algorithm eliminates bursty cleaning that is generally associated with the **chngpgs_thresh** and **softmax** database configuration parameters. If you set this registry variable to YES, the setting of the **chngpgs_thresh** configuration parameter has no effect.

Best practices

Use the following design best practices for table spaces:

- Prefer automatic storage to DMS table spaces. Automatic storage offers an important advantage with automatic container management.
- Use CIO or DIO in table spaces to bypass file system buffers and prevent double buffering, especially in databases that you migrated from Version 9.1. Ensure that the buffer pools are tuned appropriately. The result is better I/O performance. For details, see *"Table space design for OLTP workloads" on page 13*.
- Using table spaces with 8 KB or 16 KB page sizes can let you store more data on disks with lesser impact on I/O and buffer pool storage costs than 32 KB page size. If you use a larger page size and access is random, you might need to increase the size of the buffer pool to achieve the same buffer pool hit ratio for reading that you had with the smaller page size. For details, see *"Table space page sizes" on page 14*.

Use the following design best practices for buffer pools

- Create additional buffer pools for each page size used in table spaces. Having more than one buffer pool allows you to configure the memory used by the database to improve overall performance. Care must be taken in configuring additional buffer pools.
- Explicitly set the size of buffer pools or enable the STMM to tune buffer pool sizes automatically. For details, see *"Buffer pool design" on page 15*.
- Associate different buffer pools for temporary table spaces and permanent table spaces for data and large objects to avoid possible buffer pool contention. For details, see *"Buffer pool design" on page 15*.
- Set the **num_iocleaners** parameter to Automatic and the **DB2_USE_ALTERNATE_PAGE_CLEANING** registry variable to YES. For details, see *"Page cleaning activity" on page 16*.
- Monitor buffer pool usage by using the **db2pd -bufferpools** command.

Data types

Designing tables for a database involves choosing an appropriate data model and data types. Data type is a column attribute definition that indicates what type of data is stored in a table column.

Careful selection of the right data type for the nature of data stored helps minimize storage requirements.

Minimizing space consumption by data rows helps fit more rows in a data page. Having more rows in a data page improves the buffer pool hit ratio, reduces I/O cost, and achieves better query performance. DB2 supports variety of built-in data types and user-defined data types (UDTs). UDTs are extensions of the built-in data types and can be created as distinct, structured, reference, and array. For a complete list of supported data types and their descriptions, see "Data types" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.sql.ref.doc/doc/r0008483.html*.

Data type selection

When designing a physical database for OLTP workloads, selecting the appropriate data types is important. Often, abbreviated or intuitive codes are used to represent a longer value in columns, or to easily identify what the code represents. For example, an account status column whose codes are OPN, CLS, and INA represent an account that can be open, closed, or inactive. From a query processing perspective, numeric values can be processed more efficiently than character values, especially when joining values. Therefore, using a numeric data type can provide a slight benefit.

While using numeric data types might mean that interpreting the values that are stored in a column is more difficult, there are appropriate places where the definitions of numeric values can be stored for retrieval by users, such as:

- Storing the definitions as a domain value in a data modeling tool such as Rational® Data Architect, where the values can be published to a larger team by using metadata reporting.
- Storing the definition of the values in a table in a database, where the definitions can be joined to the values to provide context, such as text name or description. Tables that store values of columns and their descriptions are often referred to as reference tables or lookup tables. For large databases, storing of definitions in tables might lead to proliferation of reference tables. While this is true, if you choose to use a reference table for each column that stores a code value, you can consolidate reference tables into either a single or a few reference tables. From these consolidated reference tables, you can create virtual views to represent the lookup table for each column.

CHAR data type versus VARCHAR data type

A general guideline is that if the column length varies considerably from row-to-row, use the VARCHAR data type to minimize the space used by each row in the page.

DECFLOAT data type

The floating point data types (REAL, DOUBLE or FLOAT) represent approximation of real numbers. The DECIMAL data type represents a packed decimal number with an implicit decimal point.

For banking and financial applications where precision and accuracy of the numeric data is important, these data types might not meet the application requirement. For such applications, DECFLOAT is the right data type.

DECFLOAT represents an IEEE 754r decimal floating-point value with a decimal point. The position of the decimal point is stored in each decimal floating-point value. It can represent maximum of 34 digits. The column can be defined in two ways:

- As DECFLOAT(16) to achieve 16 digits of precision with an exponent range of 10^{-383} to 10^{+384}.
- As DECFLOAT(34) to achieve 34 digits of precision with an exponent range of or 10^{-6143} to 10^{+6144}.

Large object (LOB) data types

A large object (LOB) refers to any of the DB2 large object data types. These types are binary large object (BLOB), character large object (CLOB), and double-byte large object (DBCLOB). In a Unicode database, you can use the national character large object (NCLOB) as a synonym for DBCLOB.

The LOB data types store large unstructured data such as text, graphic images, audio, and video clips that cannot be stored in the regular character or graphic data types such as CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC.

BLOB data type versus CLOB data type

Use the BLOB data type when storing binary data that does not need to be associated with a code page. Use the CLOB data type to store large text data that must be associated with a code page and be able to be translated from one character set into another. If you store large text data in CLOBs, you can select the text in a query and have indexes on the CLOB column that can speed up query access.

Storing LOBs and inline XML columns in table rows

LOBs and XML columns are generally stored in a location separate from the table row that references them. If LOB or XML columns are accessed frequently and their data can fit within the data page with rest of the columns, storing inline data with the table row offers better performance. Storing inline date for LOB or XML columns reduces I/O and simplifies the access to the data and the manipulation of the data. You can choose to have LOB or XML data that falls below a size threshold that you specify included as inline data. These LOB or XML columns can then be manipulated as part of the base table row, which simplifies operations such as movement to and from the buffer pools. In addition, the inline data would qualify for row compression if row compression is enabled. The INLINE LENGTH option of the CREATE and ALTER TABLE statements allows LOB or XML data smaller than the specified inline length to be included in the base table row.

The following figure shows LOB descriptors within the base table row which are references to the LOBs location:

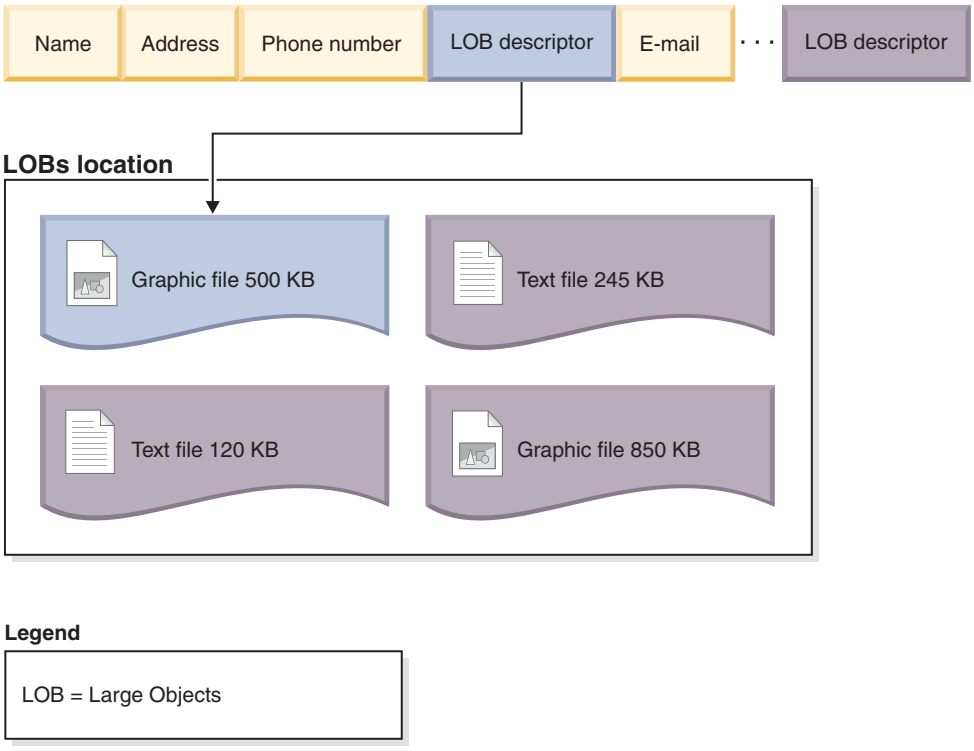

Figure 2. LOB descriptors within the base table row refer to the LOBs within the separate LOBs location

The following figure illustrates how LOBs can be included within base table rows as inline data:

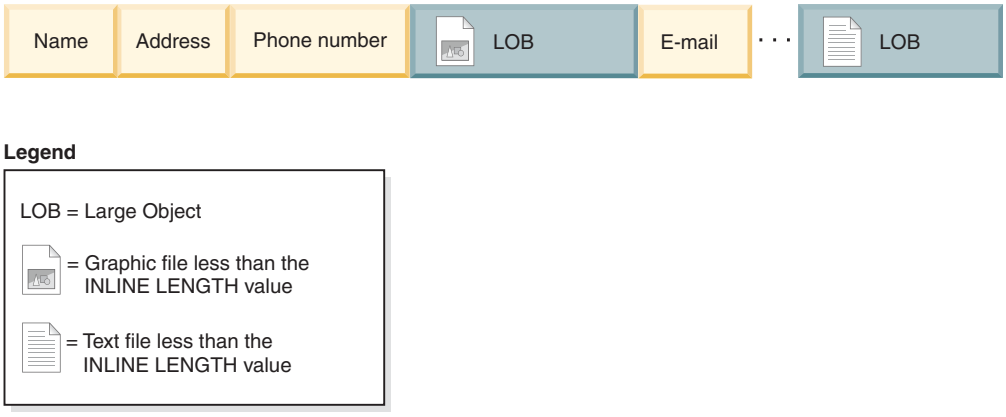

Figure 3. Small LOBs included within base table rows

Best practices

Use the following design best practices for selecting data types:

- Always try to use a numeric data type over a character data type, taking the following considerations into account:
  - When creating a column that holds a Boolean value (“YES” or “NO”), use a DECIMAL(1,0) or similar data type. Use 0 and 1 as values for the column rather than “N” or “Y”.
  - Use integers to represent codes.
  - If there are less than 10 code values for a column, the DECIMAL(1,0) data type is appropriate. If there are more than 9 code values to be stored in a column, use SMALLINT.
- Store the data definitions as a domain value in a data modeling tool, such as InfoSphere Data Architect, where the values can be published by using metadata reporting.
- Store the definition of the values in a table in a database, where the definitions can be joined to the value to provide context, such as “text name” or “description”.

Tables

DB2 databases store data in tables. There are several types of tables to store persistent data such as multidimensional clustering tables, partitioned tables, and range clustered tables. In addition to tables used to store persistent data, there are also tables that are used for presenting results, summary tables, temporary tables.

Depending on what your data is going to look like and the type of transactions, you might find a table type offers specific capabilities that can optimize storage and query performance for your environment.

Choosing the right type of table

Choosing the type of table depends on business and application requirements, nature of the data stored in the table and query performance requirements. The following section describes when each type of the table is right.

Base tables

Base tables hold persistent data. DB2 databases have the following types of base tables:

Regular tables
Regular tables with indexes are for general-purpose usage.

Range partitioned tables
Table partitioning is a data organization scheme in which table data is divided in multiple storage objects called data partitions based on one or more table partitioning key columns. Tables are partitioned by column value range and can have local index for each data partition or a global index for the entire table.

Multidimensional clustering (MDC) tables
MDC tables are physically clustered on more than one key, or dimension, at the same time. MDC tables provide guaranteed clustering within the composite dimensions.

Range-clustered (RCT) tables
RCT tables are implemented as sequential clusters of data that provide fast, direct access. At table creation time the entire range of pages is preallocated based on the record size and the maximum number of records to be stored.

The following sections describe some of these base tables in more detail and other types of tables such as temporary tables. Materialized query (MQT) tables are not discussed in this paper. MQTs are a powerful way to improve response times for complex queries and therefore are better suited for data warehousing environments.

Splitting tables

Creating a data model to store the data in multiple tables and placing the columns based on application requirements and usage would offer better performance. When you design a table, consider how the data is used by the applications and the application requirements.

In OLTP environments, splitting a large table into multiple pieces improves the query performance over queries that must scan one large table.

Consider a scenario where you want to create a table with 500 columns. However, your application touches only 50 columns out of the 500 frequently. In this scenario, creating one large table with 500 columns gives you poor performance because the large table reduces number of rows that can fit in a page. A reduced number of rows per page causes more I/O to read than reading the same set of rows from a table that contains only the frequently used columns. In addition, the buffer pool hit ratio is low, and the application reads columns that are not needed.

The range partition and multidimensional clustering tables sections describe how dividing the table and organizing the data storage into multiple pieces either by range of values, dimensions, or both can offer improved query performance by taking advantage of the benefits offered by these tables.

Range partitioned tables

Table partitioning can be used in OLTP environments for large tables to provide easier maintenance and better query performance. The DB2 optimizer performs range elimination and scans only the relevant partitions to improve the query performance. Online maintenance of range partitioned table is intended to be easier and reduce overall administration costs on large tables because of the following features:

- BACKUP, RESTORE, and RUNSTATS commands can be run at the individual table partition level.
- Table partitions can be easily rolled in and rolled out of the database.
- Flexible index placement.

As of DB2 Version 9.7, partition level reorganization makes maintenance easier, and partitioning local indexes provides better performance. Local indexes are preferred over global indexes because local indexes do not require index cleanup when a partition is attached or detached.

Use range partitioned tables under the following conditions:

- Your application requires a larger table capacity.
- Your data can be logically organized into several data partitions based on one or more column value ranges.
- Your application requires fast online roll-in and roll-out of a large range of data.
- Your business require backup and restore of individual data partitions instead of an entire table. Placing data partitions in different table spaces allows the backing up and restoring of a specific range of data.
- You want increased query performance through partition elimination and local indexes.
- Your business objectives include better data lifecycle management.

MDC tables

If you have data that has the potential for being clustered along multiple dimensions, such as a table that tracks retail sales by geographic region, division, and supplier, an MDC table might suit your purposes.

Use MDC tables under the following conditions:

- Your data is clustered and can be organized based on multiple dimensions.

- You require a guaranteed method of maintaining clustering automatically.
- You need new ranges of data to be created dynamically in their own cells as the data arrives.
- Your application requires fast online roll-in and roll-out of large ranges of data.
- You need finer granularity of load and backup operations.

MDC tables provide the benefits of clustering data across more than one dimension in a fast and automatic way. Some of these benefits are improved performance for querying multiple dimensions and reduced overhead of table reorganization and index maintenance.

Table reorganization defragments the data by eliminating unused space and reordering rows to incorporate overflow rows. You can issue the **REORG TABLE** command to reorganize tables. You specify an index with this command to reorder the data according to this specified index. Table reorganization helps to improve data access and query performance because minimizes data reads.

A potential use of MDC tables in an online transaction processing (OLTP) environment is to avoid table reorganization. A table cannot typically be used when it is being reorganized. MDC tables help avoid table reorganization by maintaining clustering.

For MDC, one of your key decisions is to decide which column or columns should serve as MDC dimensions. The design challenge is to find the best set of dimensions and granularity to maximize grouping but minimize storage requirements. Finding this set requires knowledge of the pattern of queries that will be run. Good dimension candidates are columns that have any or all of the following characteristics:

- Used for range, equality, or IN-list predicates
- Used to roll in, roll out, or other large-scale delete operations on rows
- Referenced in GROUP BY or ORDER by clauses
- Foreign key columns
- Used in the JOIN clauses in the fact table of star schema database
- Column data values have coarse granularity; that is, few distinct values

RCT tables

In single partition environments, use RCT tables when you have the following conditions:

- Data is tightly clustered, and sequence key ranges and record key are monotonically increasing
- Table do not have duplicate key values
- Storage preallocation for table is possible
- Key range in the table is permanent

RCT tables can dramatically improve performance for some workloads. Each record in an RCT table has a predetermined record ID (RID). Rows are organized in RCT tables to provide fast, direct access without indexes to a row or set of rows. This access is accomplished through sequential numeric primary key values such as an employee ID. Transaction processing applications often generate sequential numbers for use as primary key values. Such databases often benefit the most from implementing RCT tables.

One of the main considerations with RCT tables is that space for the table is preallocated and reserved for use at the table creation time based on the record size and maximum number of records.

For more details about using RCT tables and examples, see "Range-clustered tables" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.partition.doc/doc/c0011068.html*.

Temporary tables

Temporary tables are used as temporary work tables or staging area for various database operations.

Declared global temporary tables are not commonly used by customers in OLTP environments. Use declared temporary tables to potentially improve the performance of your applications. When you create a declared temporary table, the database manager does not insert an entry into the system catalog tables; therefore, your DB2 server does not suffer from catalog contention issues.

By contrast, created global temporary tables (CGTTs) appear in the system catalog and are not required to be defined in every session where they are used. As a result, the CGTT definition is persistent and can be shared with other applications across different connections. Each connection that references the CGTT has its own unique instance of the table.

In comparison to regular tables, the database manager does not lock declared temporary tables or their rows. If you specify the NOT LOGGED parameter when you create declared temporary tables, the database manager does not log declared temporary tables or their contents. If your application creates tables to process large amounts of data and drops those tables after the application finishes manipulating that data, consider using declared temporary tables instead of regular tables.

Table storage and performance

There are various options in the CREATE TABLE statement that provide additional characteristics and benefits. When designing tables, you must do the following steps:

- Determine the space requirements for tables and user data.
- Understand the data stored in the table.
- Determine whether you take advantage of certain features, such as compression and optimistic locking

To take advantage of these additional characteristics, see "Designing tables" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.dbobj.doc/doc/c0051500.html*.

Best practices

Use the following design best practices for tables:

- Use range-clustered tables for tightly clustered data to provide fast direct access to data.
- Use table partitioning to logically organize tables, improve recovery efficiency, and improve data roll-out efficiency.
- Use MDC tables to organize and cluster data based on multiple dimensions and guarantee automatic clustering.
- Partition tables with large-scale data by both range partitioning and multidimensional clustering to take advantage of data partitions and block elimination to improve query performance.
- Use the DB2 design advisor to get recommendations on the repartitioning of tables, the conversion to multidimensional clustering (MDC) tables, and the deletion of unused objects. For more details, see "The Design Advisor" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.perf.doc/doc/c0005144.html*.

Indexes

In DB2 databases, an index is a set of pointers that are logically ordered by the values of one or more keys. The pointers can refer to things such as rows in a table, blocks in an MDC table, or XML data in an XML storage object.

Indexes are typically used to speed up access to rows in a table. When good indexes are defined on table, a query can access rows faster.

Indexes are database objects. They are physical objects, not logical objects like views. Therefore, indexes also take storage space and need necessary maintenance that uses system resources. A well-designed set of indexes can improve DB2 system performance.

Types of indexes

There are many types of indexes to choose for different purposes while designing a physical DB2 database model.

Unique indexes and nonunique indexes
You can use unique indexes to enforce uniqueness on index columns of a table. If a unique index is created on a table, rows cannot have identical data values on the index key columns. Nonunique indexes do not have this constraint on the table.

Clustered and nonclustered indexes
Clustered indexes are indexes for which the order of the rows in the data pages corresponds to the order of the rows in the index. Only one clustered index can exist in a table. However, there is no practical limitation of the number of nonclustered indexes on a table.

Partitioned and nonpartitioned indexes
These types of indexes are only for range partitioned tables. A partitioned index is made up of a set of index partitions, each of which contains the index entries for a corresponding data partition. Each index partition contains references only to data in its corresponding data partition. A nonpartitioned index applies to the whole table.

XML indexes
An index-over-XML column is an XML index. An XML index uses a particular XML pattern expression to index paths and values in XML documents that are stored in a single XML column.

Multidimensional cluster (MDC) block indexes
When you create an MDC table, two indexes are created automatically: a dimension-block index that contains pointers to each occupied block for a single dimension and a composite-block index that contains all dimension key columns and is used to maintain clustering during insert and update activity.

Index guidelines for OLTP workload

Any number of indexes can be defined on a particular table, to a maximum of 32 767 indexes. They can have a beneficial effect on the performance of queries.

The index manager must maintain the indexes during delete, insert, and update operations. The major part of OLTP workloads consists of delete, insert, and update operations. Therefore, creating large index keys or many indexes for a table that receives many updates can slow down the processing of these operations.

Indexes use disk space as they are physical database objects. The amount of disk space used varies depending on the length of the key columns and the number of rows that are being indexed. The size of the index increases as more data is inserted into the table. Therefore, consider the amount of data that is being indexed when planning the size of the database.

To index or not to index

While considering an index on a table note that the benefits carry certain costs. The sole purpose of index is to speed up the lookup of a particular value from a table. Besides the cost in storage, there is an additional cost of index maintenance during delete, insert, and update operations.

When creating indexes, keep in mind that although indexes can improve read performance, they negatively impact write performance. This negative impact occurs because the database manager must update indexes for every row that the database manager writes to a table. Therefore, create indexes only when there is a clear overall performance advantage.

Good candidates for index columns

Building indexes on all primary keys (PKs) and most foreign keys (FKs) is important because most joins occur between PKs and FKs. Indexes on FKs also improve the performance of referential integrity checking. Explicitly provide an index for a PK for easier administration. If you do not specify a PK, the DB2 database manager automatically generates one with a system-generated name which is more difficult to administer.

Columns frequently referenced in WHERE, GROUP BY, or ORDER BY clauses are good candidates for an index. An exception to this rule is when the predicate provides minimal filtering. Indexes are seldom useful for inequalities because of the limited filtering provided. An example of an inequality in a WHERE clause is `WHERE cost <> 4`.

Choosing the leading columns of a composite index facilitates matching index scans. The leading columns should reflect columns frequently used in WHERE clauses. The DB2 database manager navigates only top down through a B-tree index for the leading columns used in a WHERE clause, referred to as a matching index scan. If the leading column of an index is not in a WHERE clause, the optimizer might still use the index, but the optimizer is forced to use a nonmatching index scan across the entire index.

Similarly, columns that figure in a GROUP BY clause of a frequent query might benefit from the creation of an index. These columns benefit particularly if the number of values that are used to group the rows is small relative to the number of rows that are being grouped.

Ordering the columns in an index key from the most distinct to the least distinct provides faster data access. Although the order of the columns in an index key does not make a difference in its creation, it might make a difference to the optimizer when it is deciding whether to use an index. For example, if a query has

an `ORDER BY col1,col2` clause, an index created on `(col1,col2)` could be used, but an index created on `(col2,col1)` might not be used. Similarly, if the query specified a condition such as `WHERE col1 >= 50 and col1 <= 100` or `WHERE col1=74`, then an index on `(col1)`or on `(col1,col2)`could be helpful, but an index on `(col2,col1)` is far less helpful.

Using include columns can enable index-only access for data retrieval, thus improving performance. An include column is a column that is not part of the unique index key but which is to be stored or maintained in the index. Include columns can be specified when creating unique indexes by using the CREATE INDEX statement with the INCLUDE clause. Only the unique-key columns are sorted and considered for uniqueness.

For example, if there is a unique index on col1, and col2 was specified as an include column, a query like `SELECT col1, col2 FROM table1 WHERE col1 < 10` results in index-only access.

Include columns increase index space requirements. If the included columns are updated frequently, include columns also increase index maintenance costs. The maintenance cost of updating include columns is less than the cost of updating key columns, but more than the cost of updating columns that are not part of an index.

Indexes for range partitioned tables

Partitioned indexes offer benefits when performing roll-in operations with partitioned tables. Roll-in operations consist of attaching a data partition to another table by using the ATTACH PARTITION clause on the ALTER table statement.

With a partitioned index, you can avoid the index maintenance that you must otherwise perform with nonpartitioned indexes. When a partitioned table uses a nonpartitioned index. You must use the SET INTEGRITY statement to perform index maintenance on the newly combined data partitions. Not only is this operation time consuming, it also can require a large amount of log space, depending on the number of rows that are being rolled in.

Clustering indexes

Clustering indexes incur additional overhead for insert and update operations when the row to insert or update cannot be stored on the same page as other records with similar index key values. Because an OLTP workload performs many of these operations, you need to weigh the benefits of clustering indexes for queries against the additional cost to insert and update operations. In many cases, the benefit far outweighs the cost, but not always.

For a clustering index, the database manager attempts to place new rows for the table physically close to existing rows with similar key values as defined by the index. A clustered index is most useful for columns that have range predicates because a clustered index allows better sequential access of data in the table. Sequential data access results in fewer page fetches because like values are on the same data page.

If you want a primary key index to be a clustering index, do not specify a primary key on the CREATE TABLE statement because the associated index to the primary key cannot be modified. Instead, create the table by performing the following steps:

1. Issue a CREATE TABLE without a primary key.

2. Issue a CREATE INDEX statement that specifies clustering attributes.
3. Use the ALTER TABLE statement to add a primary key that corresponds to the clustering index that you created.

This clustering index is used as the primary key index. Generally, clustering is more effectively maintained if the clustering index is unique.

Only one clustered index can be defined for each table. Therefore, choose the index that is most commonly used and define it as clustered index.

To reduce the need for frequent reorganization, when using a clustering index, specify a PCTFREE value when you create the index to leave enough free space on each index leaf page as it is created to avoid page splits. During future activity, rows can be inserted into the index with less likelihood of causing index page splits. Page splits cause index pages to not be contiguous or sequential, which in turn results in decreased efficiency of index page prefetching.

The PCTFREE value specified when you create the relational index is retained when the index is reorganized.

Reorganizing or dropping and recreating indexes relocates the indexes to a new set of pages that are roughly contiguous and sequential and improves index page prefetch. Although more costly in time and resources, the REORG TABLE utility also ensures clustering of the data pages. Clustering has greater benefit for index scans that access a significant number of data pages.

Indexes for tables with XML data

For queries on XML data, query access through indexes defined over XML columns can improve query performance.

Although you can specify multiple columns in indexes, you can specify only one XML column in indexes over XML columns. In addition, you can specify only one XML pattern for indexes over XML columns. If you want to associate more than one XML pattern with an XML column, you can specify multiple indexes over that XML column.

Adjust indexes design

OLTP workloads involve many insert and update operations. This volume of operations frequently changes the statistics of database objects. Review and adjust your indexes design to ensure that the existing indexes still influence performance positively.

Commonly used indexes

To see index use statistics for your OLTP workloads, use the **db2pd -tcbstats index** command.

The indexes are referenced by using the IID, which can be linked with the IID for the index in the SYSIBM.SYSINDEXES system catalog table. At the end of the command output is a list of index statistics. Scans indicate read access on each index, while the other indicators in the output provide insight on write and update activity to the index.

The following text is an extract of the **db2pd -tcbstats index** command:

```
TCB Index Stats: Address TableName IID PartID EmpPgDel RootSplits
BndrySplts PseuEmptPg EmPgMkdUsd Scans IxOnlyScns KeyUpdates InclUpdats
NonBndSpts PgAllocs Merges PseuDels DelClean IntNodSpl
```

DB2 Design Advisor

You can use the DB2 Design Advisor to determine which indexes are never accessed for specific workloads. Then eliminate the identified indexes and benchmark your applications to verify that the index elimination has no effect on performance.

Also, you can use the Design Advisor to determine redundant indexes. Indexes that use the same or similar columns have the following issues:

- They make query optimization more complicated.
- They use storage.
- They seriously affect the performance of insert, update, and delete operations.
- They often have marginal benefits.

An example of a redundant index is one that contains only an account number column when there is another index that contains the same account number column as its first column.

Best practices

Use the following design best practices for indexes:

- Use an index only where there is a clear advantage for frequent access exists. For details, see *"Index guidelines for OLTP workload" on page 29*.
- Use columns that best match with the most frequent used queries as index keys. For details, see *"Index guidelines for OLTP workload" on page 29*.
- Use include columns for two or more columns that are frequently accessed together to enable index only access for queries. For more details, see *"Index guidelines for OLTP workload" on page 29*.
- Use partitioned indexes for partitioned tables. For more details, see *"Indexes for range partitioned tables" on page 31*.
- Create clustered index on columns that have range predicates. Indicate a PCTFREE value to reduce the need of index reorganization. For OLTP workloads with significant insert or update operations, use a large PCTFREE value. For more details, see *"Clustering indexes" on page 31*.
- Create indexes over XML columns for faster performance on queries over XML data. For more details, see *"Indexes for tables with XML data" on page 32*.
- Use the Design Advisor to get recommendations for RID-based indexes on MDC tables or MDC dimensions for a table. Choosing the right dimensions for your OLTP workload is important because block indexes are automatically created for each dimension. For more details, see *"Indexes for tables with XML data" on page 32*.
- Eliminate indexes that are never accessed. Use Design Advisor to determine whether you have indexes that have not been accessed. For more details, see *"Adjust indexes design" on page 32*.
- Avoid redundant indexes. Use Design Advisor to determine whether you have redundant indexes. For more details, see *"Adjust indexes design" on page 32*.

Database transaction logs

Database transaction logging is crucial for database recovery and an important part of designing a highly available database solution.

Database logs make it possible to recover from a failure. They also make it possible to synchronize primary and standby databases in HADR environments.

DB2 uses a separate set of log files for each database.

All databases have logs associated with them. These logs keep records of database changes. If a database needs to be restored to a point beyond the last full, offline backup, logs are required to roll the data forward to the point of failure. DB2 databases support two types of database logging: circular logging and archive logging.

Circular logging

Circular logging supports only crash recovery, that is if a DB2 instance crashes for some reasons such as power failure or user error, the next database restart uses information from the log files to bring database to a consistent point.

During crash recovery all closed, committed or aborted, transactions that were not written to disk are written to disk. All open transactions , not yet committed, are rolled-back to remove partial changes.

In development and test environments, you could use circular logging. To simplify database administration in these environments where transaction logging is not essential, use circular logging.

Archive logging

The advantage of choosing archive logging is that rollforward recovery can use both archived logs and active logs to restore a database either to the end of the logs or to a specific point in time.

The archived log files can be used to recover changes made after a database backup was taken. This type of logging is different from circular logging where you can recover only to the time of the backup, and all changes made after that are lost.

When a database is created, the default logging type is circular logging. Update the **logarchmeth1** or the **logarchmeth2** database configuration parameter to enable archive logging. After changing the parameter setting, perform an offline database backup to be able to access the database.

Configuring transaction logging

DB2 supports several media types for log archiving such as, DISK, TSM (for Tivoli® Storage Manager support, VENDOR (for third-party library support), or a customer program that uses USEREXIT settings. You can set the **logarchmeth1** database configuration parameter to any possible valid value. However, the **logarchmeth2** database configuration parameter supports only the OFF, DISK, TSM, and VENDOR values.

The nature of OLTP workload puts heavy demands on the response time and throughput of transaction logging devices. For best performance and availability reason, transaction logs should be placed on dedicated, faster devices, and in separate file systems. For improved performance, do not share transaction log file systems or I/O bandwidth with any other database storage object such as table spaces.

By default, the directory for transaction logs is set to the database directory. Change it by setting the **newlogpath** database configuration parameter.

To prevent a rogue application from using up all the transaction log space and affecting the database, consider by using the following database configuration parameters for transaction logging:

- The **num_log_span** parameter specifies whether there is a limit to how many log files one transaction can span and what that limit is.
- The **max_log** parameter specifies whether there is a limit to the percentage of log space that a transaction can consume and what that limit is.
- The **blk_log_dsk_ful** parameter prevents log space full errors from being generated when the DB2 database manager cannot create a new log file in the active log path. Instead of generating a disk full error, the DB2 database manager attempts to create the log file every 5 minutes until it succeeds.

You can use the logging dashboard and the alert thresholds in IBM InfoSphere Optim™ Performance Manager to monitor the log space utilization and determine whether you need to change the logging space configuration. Fore more details, see "Monitoring with Optim Performance Manager" at *http://publib.boulder.ibm.com/infocenter/perfmgmt/v5r1/topic/com.ibm.datatools.perfmgmt.monitor.doc/p_monitor.html*.

Mirror log path

You can set an alternate path for transaction logging by using the **mirrorlogpath** database configuration parameter. If this parameter is set, the database manager creates active log files in both the log path and the mirror log path. All log data is written to both paths, increasing protection from accidental loss of a log file.

To get maximum benefits of having a mirror log path without performance degradation, we recommend that the use of dedicated fast devices for the mirror log path. Placing the mirror log path on the same device or file system as the log path can cause the I/O bandwidth become a bottleneck. Also, it will protect you from only a few scenarios, but will not protect from total loss of device or filesystem.

To get maximum benefits of having a mirror log path without performance degradation, use dedicated fast devices for the mirror log path. Placing the mirror log path on the same device or file system as the log path can cause the I/O

bandwidth become a bottleneck. Also, it protects you from only a few scenarios, but does not protect you from the total loss of a device or file system.

If you want to use a mirror log path, factor in the requirements in your estimate. You need double the storage space for transaction logging and increased I/O bandwidth.

Data and index compression

Data and index compression might help minimize the size of transaction logging. If you use DB2 compression capabilities, user data written to log records as a result of INSERT, UPDATE, and DELETE activity is smaller. However, it is possible that some UPDATE log records are larger when compressed than when not using compression.

Besides compression, you can do other things to minimize log space consumption. Minimize log space consumption by grouping together columns that are updated more frequently and place them at or near the end of the record definition. For more details, see *"Data and index compression" on page 39*.

Even with a good compression ratio, OLTP workloads do not benefit from better transaction response time or throughput. Because data and index compression can reduce table space I/O and logging I/O, administration tasks such as database backups and archiving logs can can be performed in considerable less time on compressed data.

Best practices

Use the following design best practices for database logging:

- Use archive logging in production environments to be able to perform many recovery operations including, online backup, incremental backup, online restore, point-in-time rollforward, and issuing the RECOVER DATABASE command.
- Consider enabling the **trackmod** database configuration parameter for incremental backups to track database modifications so that the **BACKUP DATABASE** command can determine which subsets of database pages should be included in the backup image for either database backups or table space backups.
- Use mirror log path to improve high availability. For more details, see *"Mirror log path" on page 36*.
- Configure secondary log files to provide additional log space on a temporary basis.
- Use data and index compression to improve performance of administrations tasks such as database and table space backups because the backups on compressed data take less time.
- Consider the I/O adapter or bus bandwidth requirements for transaction logging. If requirements are not adequate, I/O usage might result in a bottleneck. If high availability is a concern, look into operating system level support for I/O multi-pathing.

Data and index compression

You can reduce the amount of storage needed for your data by using the compression capabilities built into DB2 for Linux, UNIX, and Windows databases to reduce the size of your tables, indexes, and backup images.

Tables and indexes often contain repeated information. This repetition can range from individual or combined column values, to common prefixes for column values, or to repeating patterns in XML data. Compression methods can use short strings or symbols to replace repeated information.

There are a number of compression capabilities that you can use to reduce the amount of space required to store your tables and indexes, along with features you can employ to determine the savings compression can offer. You can also use backup compression to reduce the size of your backups.

The compression capabilities included with most editions of DB2 Version 9.7 include:

- Value compression
- Backup compression

The following additional compression capabilities are available with a license for the DB2 Storage Optimization Feature:

- Row compression, including compression for XML storage objects
- Temporary table compression
- Index compression on compressed temporary or permanent tables

Row compression

Row compression, sometimes referred to as static compression, compresses data rows by replacing patterns of values that repeat across rows with shorter symbol strings.

The main benefit of using row compression is that you can store data in less space, which can yield significant savings in storage costs. Also, because you use storage at a slower rate, future expenditures for additional storage can be delayed.

In addition to the cost savings, row compression can improve performance. Many queries against compressed data can be performed with fewer I/O operations because each read from disk brings in more data. Similarly, more data can be cached in the buffer pool, increasing buffer pool hit ratios. However, there is a trade-off in the form of the extra CPU cycles that are needed to compress and decompress data.

There are two requirements to make a table ready for row compression:

- You must make a table eligible for compression by creating or altering a table with the COMPRESS YES clause.
- You must build a dictionary of values or symbols from the table that will be compressed. Depending on the release of the DB2 database product you use, there are different means to build the compression dictionary.

After these two requirements are satisfied, data inserted or updated in the table can be compressed.

Here is a simple example of enabling row compression in a table and index:

```
CREATE TABLE T1 (C1 INT, C2 INT, C3 INT) COMPRESS YES
CREATE INDEX I1 ON T1(C1)
SELECT COMPRESSION FROM SYSCAT.INDEXES WHERE TABNAME='T1'

 COMPRESSION
 -----------
 Y

    1 record(s) selected.
```

Good candidates for row compression

Examine your database to determine which tables within the database might be candidates for compression. Initially, you enable data compression to save storage on existing uncompressed tables. Later, data compression helps you optimizing future storage growth. Your storage "pain points" can be found in existing tables within your database, in tables where you anticipate increased growth over time, or both.

Your largest tables are obvious candidates for row compression, but do not overlook smaller tables. If you have hundreds or thousands of smaller tables, you might benefit from the aggregate effect of compressing many smaller tables. Large and small are relative terms here. Your database design determines whether tables of a million or several million rows are large or small.

Small tables under a few hundred KB in size are not good candidates for row compression when the space savings that can be achieved are not offset by the storage requirements of the data compression dictionary. The size of dictionary for a small table can be approximately 100 KB and is stored in the physical table data object. As a rule of thumb, consider compressing small tables which are 2 MB in size or larger.

Read-only tables are great candidates for compression. If the table experiences only some updates, it is likely to be a good candidate. Tables that undergo heavy updates might not be as good candidates for compression. Tables with a read/write ratio of 70/30 or higher, are excellent candidates for compression.

Separate large tables into their own table space before attempting to compress those tables.

Estimates for compression ratios

For versions earlier than DB2 Version 9.5, use the following commands to estimate compression for tables:

```
$ DB2 INSPECT ROWCOMPESTIMATE TABLE NAME table_name RESULTS KEEP file_name
$ db2inspf file_name output_file_name
```

For DB2 Version 9.5, use the following table function to estimate compression for tables:

```
SELECT * FROM TABLE(SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO
                              ('schema','table_name','ESTIMATE')) AS T
```

For DB2 Version 9.7, use the following table function to estimate compression for tables:

```
SELECT * FROM TABLE(SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO_V97
                                ('schema','table_name','ESTIMATE')) AS T
```

Building a compression dictionary

Starting with DB2 Version 9.5, automatic compression dictionary creation (ADC) and classic table reorganization are the two primary methods to build table-level compression dictionaries.

The quality of a compression dictionary is based on the data used to create it. When performing a classic reorganization to compress a table, all the table data is sampled as part of the dictionary building process. Therefore, you should reorganize your table after it contains a good representation of sample data.

Instead of reorganizing a table offline to create a compression dictionary, you can create a smaller copy of the table. This smaller table can then be reorganized and compressed. The data from the original table can then be loaded into this new version of the table and that data is compressed with the dictionary that was built. This method avoids the need to completely reorganize a table in order to compress a table.

For DB2 databases in Version 9.5 or later releases, a compression dictionary is created automatically if each of the following conditions is met:

- You set the COMPRESS attribute for the table to YES. You can set this attribute to YES when you create the table, by using the COMPRESS YES option of the CREATE TABLE statement. You can also alter an existing table to use compression by using the same option on the ALTER TABLE statement.
- A compression dictionary does not exist for that table.
- The table reaches a size such that there is sufficient data to use for constructing a dictionary of repeated data. This size is usually on the threshold of 2 MB.

With ADC, an offline reorganization of the table or the running of the INSPECT utility is not required to create the compression dictionary.

The following diagram shows the process by which the compression dictionary is automatically created:

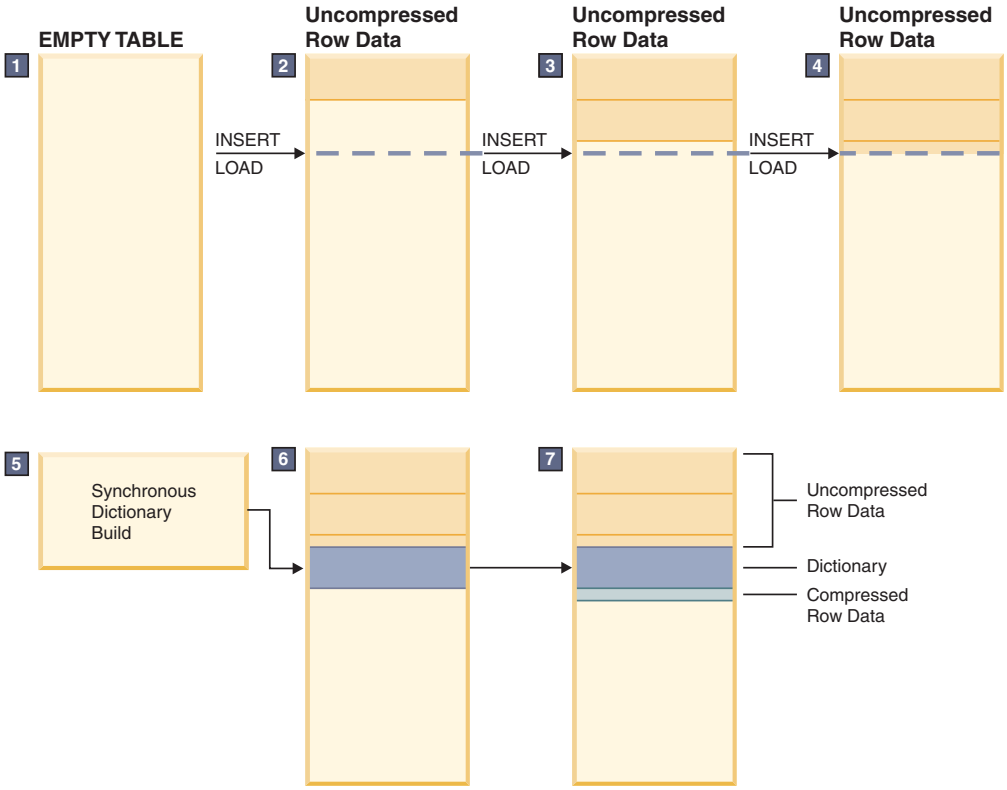

Figure 4. Automatic creation of compression dictionary

Another alternative method to the primary methods is to use the INSPECT command to create a compression dictionary and insert it into a table. If a compression dictionary does not exist for a table, the INSPECT command inserts it in the table.

To use this method, use the ALTER TABLE statement to set the table COMPRESS attribute to YES and then issue the INSPECT ROWCOMPESTIMATE command. During this operation, the table remains online. After the compression dictionary exists in the table, all subsequent data that is added is subject to compression but existing rows remain uncompressed.

Clustering data during table reorganization

If you want to reorganize your table and recluster it according to a particular index, the REORG utility can accomplish this goal by applying a scan-sort or an index-scan method.

The database manager uses the scan-sort method by default when an index is specified in the REORG command. This method involves scanning the table and sorting the results in memory. Although, the sort might spill to disk through a temporary table space.

The index-scan REORG method requires the explicit use of the INDEXSCAN keyword. It does not use a sort because it follows the order of the index and fetches the records in the original table for each RID in the index.

A REORG TABLE operation on a table without a clustering index and a REORG TABLE operation through index-scan processing, requires an extra pass of the table in order to build the compression dictionary, while scan-sort processing does not. Therefore, the default REORG scan-sort processing can typically complete the reorganization and dictionary creation quicker.

If the table is sparsely populated, then a reclustering REORG TABLE operation that uses index-scan processing is faster than scan-sort processing. Moreover, an index-scan REORG TABLE operation can be beneficial when indexes are highly clustered and there is not enough memory and temporary space available in the system into which a scan-sort would spill.

Do not specify an index on the REORG command when compressing tables unless reclustering is required. By not specifying an index in this situation, you avoid the extra processing and sort resources it takes to recluster the data. You also minimize the time it takes to perform the data compression.

Index compression

Index objects and indexes on compressed temporary tables can also be compressed to reduce storage costs. Compressing these objects is useful for large OLTP and data warehouse environments where it is common to have many large indexes.

By default, index compression is enabled for compressed tables, and disabled for uncompressed tables. You can override this default behavior by using the COMPRESS YES option of the CREATE INDEX statement. When working with existing indexes, use the ALTER INDEX statement to enable or disable index compression. You must then perform an index reorganization to rebuild the index.

The following restrictions apply to index compression:

- MDC block indexes and XML path indexes cannot be compressed.
- Index specifications cannot be compressed.
- Compression attributes for indexes on temporary tables cannot be altered with the ALTER INDEX command.

Index compression can provide significant performance improvements in I/O-bound environments and maintain performance in CPU-bound environments. Consider the I/O and CPU trade-offs before you disable index compression.

Best practices

Use the following design best practices for data and index compression:

- For row compression:
  - Identify tables that are good candidates for row compression. Understanding typical SQL activity against the table can help you determine whether it is a good candidate for row compression. For more details, see *"Good candidates for row compression" on page 40*.
  - Determine which tables to compress by using compression ratio estimates. For more details, see *"Estimates for compression ratios" on page 40*.
  - Use the INSPECT command or the ADC to build a compression dictionary instead of reorganizing the table. For more details, see *"Building a compression dictionary" on page 41*.
  - If you are reorganizing a table to compress it, do not specify an index to cluster the data. Eliminating the extra processing for clustering minimizes the time it takes to perform the compression. For more details, see *"Clustering data during table reorganization" on page 42*.
  - If you are using row compression and storage optimization is of higher priority than the extra time it takes to perform a compressed backup, use backup compression. For more details, see "Backup compression" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.ha.doc/doc/c0056483.html*.
- Keep index compression enabled. For more details, see *"Index compression" on page 43*.
- Use value compression in table with many column values equal to the system default values or NULL. Value compression can be combined with other compression methods. For more details, see "Value compression" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.dbobj.doc/doc/c0056482.html*

For more details about best practices for compression, see "Best Practices: Deep Compression" at *http://www.ibm.com/developerworks/data/bestpractices/deepcompression/*.

Query design

At a fundamental level, SQL operations including select, insert, update, and delete, are the way for applications to interact with DB2 databases. Overall application performance and experience are influenced by SQL operations used by the application.

A thorough treatment of designing, maintaining, monitoring, and tuning SQL queries is beyond the scope of this paper. However, we present a high-level overview of tools and general guidelines for query design, since query design and physical database design are closely related to each other.

Most of the physical database design characteristics are not apparent to the SQL statements, but to better use DB2 features, queries need to be written with the physical characteristics of a database, such as indexes, in mind. For example, when using range partitioned table, the select query works correctly even if it does not contain a predicate with the range partitioning key. However, it might not have performance benefit of partition elimination.

On other hand, if an SQL operation fails to meet business service level agreement due to data growth or any other reason, a change in physical database design might be necessary. Examples of physical database design changes include: adding indexes, converting regular tables to range partitioned tables, or changing buffer pool size or association to achieve expected performance goals.

OLTP workload queries

Queries in OLTP workloads are normally short, involve few tables, and return small result sets. However, there are more concurrent queries in OLTP workload compared to other type of workloads.

For OLTP workloads, designing queries that return results quickly is essential to a good performance.

Also, take into account that there are typically high volumes of queries that run concurrently in an OLTP workload system. Deadlocks, rollbacks due to timeout waiting for locks, or even “hanging”-like transactions can frequently occur. A query design that leads to fewer deadlocks and rollbacks can make a significant difference in query performance.

OLTP applications are good candidates to use index scans with range delimiting predicates, because they tend to return only a few rows that are qualified by using an equality predicate against a key column. If your OLTP single queries are using a table scan, you might want to analyze the explain facility data to determine the reasons why an index scan was not used.

Isolation levels

The DB2 database manager supports the following four types of isolation levels. They are listed in decreasing order of performance impact, but in increasing order of care required when accessing and updating data.

Repeatable read (RR)
The RR isolation level locks all the rows an application references within a unit of work. With this isolation level, lost updates, access to uncommitted data, and phantom rows are not possible.

Read stability (RS)
The RS isolation level locks only those rows that an application retrieves within a unit of work.

Cursor stability (CS)
The CS isolation level locks any row accessed by a transaction of an application while the cursor is positioned on the row.

Uncommitted read (UR)
The UR isolation level allows an application to access uncommitted changes of other transactions. Uncommitted read works differently for read-only and updatable cursors.

Application deadlocks

Deadlocks impact database system performance. When a deadlock occurs, the database manager chooses what transactions (victims) to stop or to roll back. This impact would result in a bad experience for the user. If database configuration parameters are not set correctly, users might experience hanging in a deadlock situation and eventually the database administrator might need to resolve the deadlock.

When you close a cursor by issuing the CLOSE CURSOR statement with the WITH RELEASE clause, the database manager attempts to release all read locks held for the cursor. Table read locks are IS, S, and U table locks. Row-read locks are S, NS, and U row locks. Block-read locks are IS, S, and U block locks.

The WITH RELEASE clause has no effect on cursors that are operating under the CS or UR isolation levels. For cursors that are operating under the RS or RR isolation levels, the WITH RELEASE clause cancels some of the guarantees of those isolation levels. Specifically, an RS cursor might experience the nonrepeatable read phenomenon, and an RR cursor might experience either a nonrepeatable read or a phantom read.

If you reopen a cursor after closing it with the WITH RELEASE clause that was operating originally under RR or RS isolation levels, new read locks are acquired.

In some situations, locks remain after the result set is closed and the transaction is committed. Closing a CURSOR WITH HOLD before issuing a COMMIT statement ensures that locks are released. Catalog locks are acquired even in uncommitted read applications using dynamic SQL. To release catalog locks, explicitly issue the COMMIT statement.

The LOCK TABLE statement locks an entire table. Only the table specified in the LOCK TABLE statement is locked. Parent and dependent tables of the specified table are not locked. The lock is not released until the unit of work is committed or rolled back. You can use this statement to prevent lock escalation. You must determine whether locking the entire table and related tables is necessary to achieve the wanted result in terms of concurrency and performance.

Performance and monitoring

DB2 database products provide tools for query performance and monitoring such as the DB2 explain facility, RUNSTATS command, automatic statistics collection, and event monitoring.

DB2 explain facility

The DB2 explain facility provides detailed information about the access plan that the optimizer chooses for an SQL or XQuery statement.

The SQL or XQuery compiler can capture information about the access plan and the environment of static or dynamic SQL and XQuery statements. The captured information helps you understand how individual SQL or XQuery statements are executed so that you can tune the statements and your database manager configuration to improve performance.

For dynamic SQL or XQuery statements, information is collected when the application is run for the first time. For static SQL and XQuery statements, the information is collected during package bind time.

The explain facility captures the following information:

- Sequence of operations to process the query
- Cost information
- Predicates and selectivity estimates for each predicate
- Statistics for all objects referenced in the SQL or XQuery statement at the time that the explain information is captured
- Values for the host variables, parameter markers, or special registers used to reoptimize the SQL or XQuery statement

The primary use of explain information is to analyze the access paths for query statements.

Run the explain facility before and after you perform actions that affect query performance such as adjusting configuration parameters, adding table space containers, updating catalog statistics, recreating indexes, and reorganizing indexes.

Having indexes can significantly benefit query performance. If a table scan rather than an index scan was used, try reorganizing the table and indexes. Then recompile the query and use the explain tool to check whether the access plan is changed successfully.

Catalog statistics

Current statistics on tables and indexes help the DB2 optimizer choose the best access plan. Deciding which statistics to collect for a specific workload is complex, and keeping these statistics up-to-date is time-consuming.

By default, automatic statistics collection is enabled in DB2 databases. The database manager determines whether catalog statistics need to be updated. Automatic statistics collection can occur synchronously at statement compilation time by using the real-time statistics feature, or the **RUNSTATS** command can be enabled to run in the background for asynchronous collection.

Customize automatic statistics collection to your needs by setting up a statistics profile. Use the WITH DISTRIBUTION AND SAMPLED DETAILED INDEXES ALL options to minimize the impact in your system. Distribution statistics make the optimizer aware of data skew. Sample statistics reduces resource consumption during statistics collection. Detailed index statistics provide more details about the I/O required to fetch data pages when the table is accessed using a particular index. For more details about automatic statistics collection and using statistics profiles, see "Automatic statistics collection" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.perf.doc/doc/c0011762.html*.

DB2 event monitors

Event monitors can be used to keep close eye on DB2 system performance. Event monitors return information for the event types specified in the CREATE EVENT MONITOR statement. For each event type, monitoring information is collected at a certain point in time.

You can use the CREATE EVENT MONITOR FOR LOCKING statement to monitor detail deadlocks, lock timeout, and lock waits. Address these issues in a timely way to avoid downgrade of query performance in your OLTP environment.

You can use the CREATE EVENT MONITOR FOR UNIT OF WORK statement to monitor transaction events. The unit of work event monitor records an event whenever a unit of work is completed by a commit or a rollback. Also, it collects a listing of packages used within a unit of work and the nesting level at which it was used to facilitate stored procedure troubleshooting.

SQL administrative routines

Starting with DB2 Version 9.7, you can use monitor table functions to collect and view data for systems, activities, or data objects.

You can retrieve information about locks by using table functions. Unlike request, activity, or data object monitor elements, information about locks is always available from the database manager. You do not need to enable the collection of this information.

For more details about these functions, see "Monitor table functions" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.mon.doc/doc/c0055388.html*.

Optimization classes

At a given optimization class, the query compilation time and resource consumption is primarily influenced by the complexity of the query, particularly the number of joins and subqueries. However, compilation time and resource usage are also affected by the amount of optimization performed.

OLTP queries are dynamic and usually returns in a short amount of time. A low optimization class can save system resources without affecting the query performance.

Best practices

Use the following design best practices for SQL queries:

- For single queries:
  - **Use index scans rather than table scans.** If a table scan was used, try reorganizing the table and index. After recompiling the query, determine whether the access plan uses an index scan with the explain facility.
  - **Avoid complex expressions in search conditions.** Such expressions can prevent the optimizer from using catalog statistics to estimate an accurate selectivity. Also, they might limit the choices of access plans that can be used to apply the predicate.
  - **Use join predicates on expressions to promote the use of nested loop join method.** Joins that involve only a few rows, such as OLTP queries, typically run faster with nested-loop joins.
  - **Create indexes on columns that are used to join tables in a query.** Indexes on these columns can speed up local predicates.
  - **Avoid data type mismatches on join columns.** In some cases, data type mismatches prevent the use of hash joins.
  - **Do not use no-op expressions in predicates to change the optimizer estimate.** A "no-op" expression of the form `COALESCE(X, X) = X` introduces an estimation error into the planning of any query.
  - **Avoid non-equality join predicates.** Avoid these predicates because the join method is limited to nested loop.
- For multiple concurrent queries:
  - **Use the minimum isolation level that satisfies your application needs.** The less restrictive isolation level, the fewer locks required, and the less memory and CPU resources are consumed.
  - **Conditions that might cause lock escalations that should be avoided.** Lock escalations reduce concurrency and consume system resources.
- Prevent application deadlocks by:
  - Closing cursors to release the locks that they hold.
  - Closing a CURSOR WITH HOLD before issuing a COMMIT statement.
  - Using the LOCK TABLE statement appropriately.

  For more details, see *"Application deadlocks" on page 46*.
- If your application uses complex SQL requests, use DB2 parallelism on symmetric multiprocessor (SMP) computers. In general, OLTP environments do not benefit from enabling DB2 parallelism on SMP computers.
- Maintain current statistics for tables and indexes. Update statistics regularly on any critical tables, including system catalog tables. For more details, see *"Catalog statistics" on page 47*.
- Use the explain facility to monitor and evaluate query performance changes.
- Use event monitors to monitor deadlocks, locking, and units of work. For more details, see *"DB2 event monitors" on page 48*.
- Use a low optimization class such as 0 or 1 for queries with a run time of less than one second. Use a higher optimization class such as 3, 5, or 7 for longer running queries that take more than 30 seconds. For more details, see *"Optimization classes" on page 48*.

Database sizing and capacity management

Database sizing and capacity planning consist of estimating system resources required to meet the enterprise level business objectives. Good capacity planning not only focuses on meeting the current needs, but sizes the system for future requirements so that the database infrastructure scales seamlessly as business needs change. In addition, a good plan considers workload variability.

The main goal of capacity planning is to identify the system resources requirements and to design a balanced system that optimizes the resources to achieve the best performance and throughput in accordance with the business requirements.

When discussing database sizing and capacity management, the DB2 database server term includes the hardware infrastructure, I/O subsystems, and operating system required to host and support one or more DB2 instances and one or more databases.

Capacity planning can help you with the following areas:

- Developing a proposal for a set of system resources or upgrades required for the deployment early in the cycle. This proposal can help you budget for the expenses ahead of time.
- Planning and designing database systems for future business requirements ahead of reaching performance thresholds and bottlenecks.

Providing additional resources when a crisis happens uses manpower and other resources. This situation leads to more expensive solutions and lower operational efficiency in your business.

Proactively managing your system needs minimizes critical incidents and business downtime. For a 24x7 mission-critical business, any downtime directly impacts revenue, sales, client satisfaction, and business reputation.

Estimating system resources and designing a balanced system

The best method to estimate resource consumption by workload and estimate the capacity to meet business requirements is to benchmark the workloads on quality assurance or test environments. Running a fresh benchmark is a time-consuming operation. It requires manpower, system setup, management support, and commitment from various teams. Instead, you can also use past data for benchmark results or workload resource consumption to estimate the resource requirements for current deployment. But the results would not be as accurate as fresh benchmark tests.

A comprehensive process for capacity planning includes all of the following steps:

1. Gather business objectives, service level requirements, and stakeholders feedback.
2. Identify current system resources capacity such as the number of CPUs, LPARs, memory on each LPAR or system, I/O performance characteristics, storage devices, space availability, and network bandwidth.
3. Investigate the nature of the workload such as which part of the workload uses most resources and when.

4. Estimate the growth rate of workloads and their resource consumption in peak periods as business grows.
5. With the data collected from the preceding steps, estimate and size the system requirements. Run workloads on nonproduction systems to validate the estimated sizing and plan the capacity needed accurately.
6. Tune the system and database as needed based on the metrics collected from the tests.
7. Develop a capacity planning proposal and validate it. Revise the sizing and proposal and repeat earlier steps as needed.
8. Achieve consensus and approval for the capacity planning proposal from all stakeholders.
9. Implement the approved capacity plan.
10. Monitor your system performance and revise capacity planning as your business and workloads change.

The following diagram shows the workflow for the capacity planning steps described in the previous paragraph:

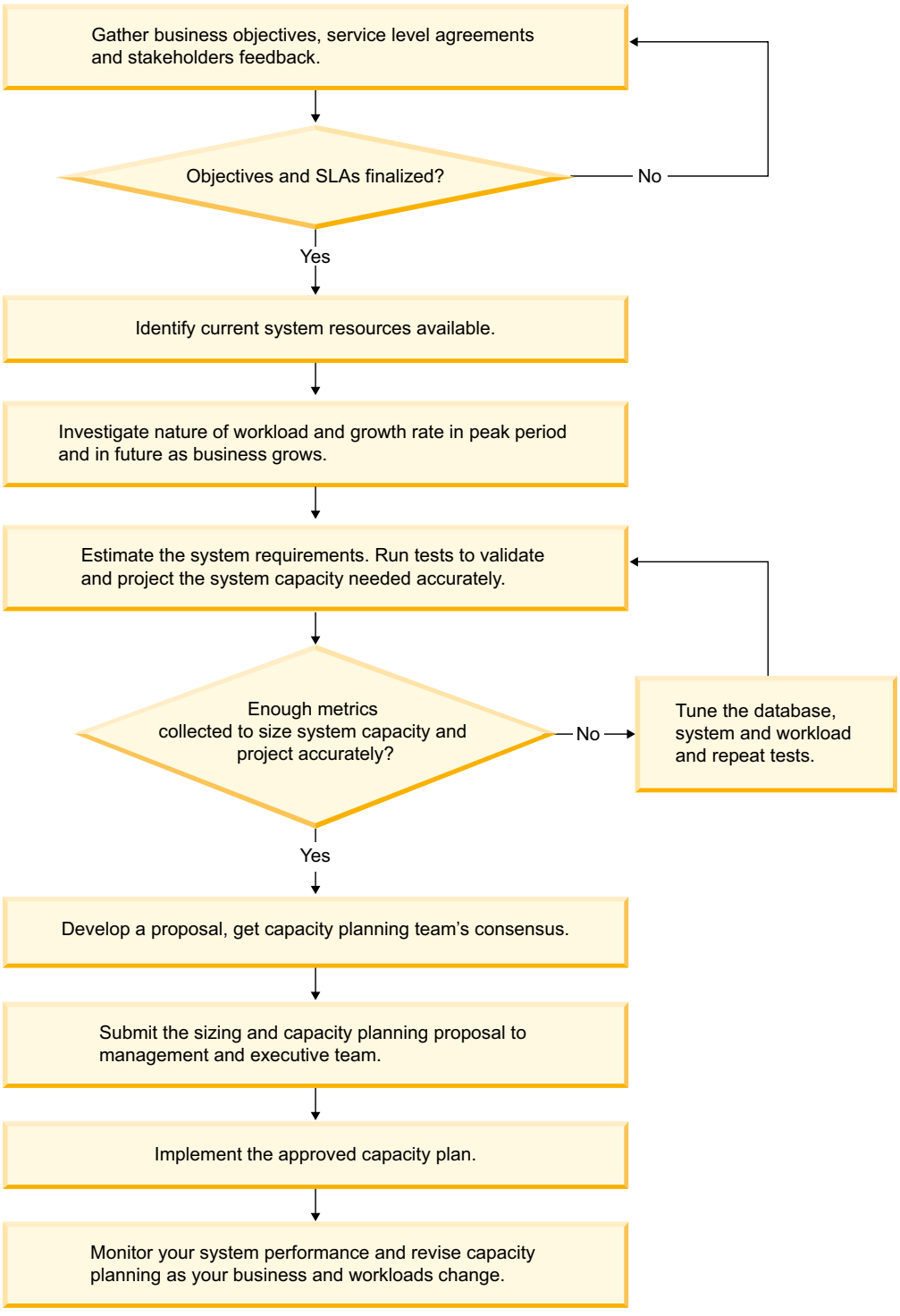

Figure 5. Process to estimate system resources and design a balanced system

When you are setting goals for throughput and response times for business applications, you must estimate and analyze all aspects of systems resources and workload for their ability to meet the service level requirements.

Investigate the nature of the workload

The answers to the following questions can help you determine the nature of the workload:

- What is the read/write ratio of the workload?
- How many read and write transactions are performed per second?
- How much data is changed per write transaction? In other words, how much transactional log data does each transaction generate?
- What is the concurrency nature of the workload? Which concurrent applications update or read data from the same tables or different ones?

Sizing system resources

System resources must be sized and designed in such way that competition among applications for system resources does not cause application performance bottlenecks. All applications that run in a business infrastructure use the following resources:

- CPU
- Memory
- Instance memory
- Number of databases hosted on the system
- I/O subsystem
- I/O performance test - read/write metrics
- Storage
- Data and transaction logging space.
- Network bandwidth

Data and transaction logs must be stored in separate disks. Size the logging disk space required based on number of tables, amount of log generation, number of active logs estimated for the workload.

Estimating memory requirements for a DB2 database server

Estimating the memory requirements for a DB2 database server for real-world workloads can be a complex task. Fortunately, the STMM and the DB2 configuration advisor make estimating memory requirements easier.

Self-tuning memory manager (STMM)

The STMM simplifies the task of memory configuration by automatically setting values for several critical memory configuration parameters. You can use the STMM to help you determine the current or future memory requirements by running different workloads for real or hypothetical scenarios.

When the STMM is enabled, the memory tuner dynamically distributes available memory resources among several memory consumers, including sort heap, package cache, lock list, and buffer pools. The feature works by iteratively modifying the memory configuration in small increments with the goal of improving overall system performance. Self-tuning memory is enabled through the **self_tuning_mem** database configuration parameter.

The following memory-related database configuration parameters can be automatically tuned:

- **database_memory** - Database shared memory size
- **locklist** - Maximum storage for lock list
- **maxlocks** - Maximum percent of lock list before escalation
- **pckcachesz** - Package cache size
- **sheapthres_shr** - Sort heap threshold for shared sorts

For database manager and database configuration parameters that are not controlled by the STMM, these configuration parameters can be tuned by using the DB2 configuration advisor.

For details about how to enable the STMM and automatically setting values for memory configuration parameters, see "Self-tuning memory overview" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.perf.doc/doc/c0024366.html*.

DB2 Configuration Advisor

The Configuration Advisor helps you tune the performance and balance the memory requirements per instance for a single database by suggesting which configuration parameters to modify and suggesting values for them. DB2 database autonomic capability runs the AUTOCONFIGURE command at database creation time. The AUTOCONFIGURE command automatically sets many of the database manager and database parameters for you. Existing databases can be tuned with either the AUTOCONFIGURE command or the InfoSphere Optim tools. The AUTOCONFIGURE command suggests the optimum value for database manager and database parameters based on your input and environment characteristics that are automatically detected.

Your input values to the AUTOCONFIGURE command and the explanation are shown in the following table:

Table 1. User input values for the AUTOCONFIGURE command

Keyword	Valid Value Range	Default Value	Explanation
mem_percent	1-100	25	Percentage of memory to dedicate to DB2 databases as derived from parameter: **instance_memory**
workload_type	Simple, Mixed, Complex	Mixed	Online Transaction Processing (OLTP) I/O intensive, Data warehousing CPU intensive
num_stmts	1-1, 000, 000	10	Number of statements per unit of work
tpm	1-200, 000	60	Transactions per minute
admin_priority	Performance, Recovery, Both	Both	Optimize for more transactions per minute or better recovery time
is_populated	Yes, No	Yes	Is the database populated with data?
num_local_apps	0-5,000	0	Number of local connected applications
num_remote_apps	0-5,000	10	Number of remote connected applications

Table 1. User input values for the AUTOCONFIGURE command (continued)

Keyword	Valid Value Range	Default Value	Explanation
Isolation	RR, RS, CS, UR	RR	Isolation Level of applications
bp_resizable	Yes, No	Yes	Are buffer pools resizable?

The configuration advisor automatically detects the following system characteristics as shown in the following table:

Table 2. System characteristics detected by the configuration advisor

Environment	Autonomically Detected System Characteristics
System	Number of physical disks and spindles
	Physical memory size
	CPU information (number of online and configured CPUs)
	OS features (OS type and release - Linux, UNIX, Windows)
Database	Size of database
	Number of tables
	Number of Indexes
	Number of table spaces
Buffer pool	Name, size, and page size for each buffer pool
	Number of buffer pools

The following text shows a sample output from the configuration advisor:

```
       Former and Applied Values for Database Manager Configuration
Description                                  Parameter   Current Value   Recommended Value
------------------------------------------------------------------------------------------
Application support layer heap size (4KB)   (ASLHEAPSZ) = 15              15
No. of int. communication buffers(4KB)(FCM_NUM_BUFFERS) = AUTOMATIC       AUTOMATIC
Enable intra-partition parallelism     (INTRA_PARALLEL) = NO              NO
Maximum query degree of parallelism   (MAX_QUERYDEGREE) = ANY             1
Agent pool size                        (NUM_POOLAGENTS) = 100(calculated) 200
Initial number of agents in pool       (NUM_INITAGENTS) = 0               0
Max requester I/O block size (bytes)         (RQRIOBLK) = 32767           32767
Sort heap threshold (4KB)                 (SHEAPTHRES) = 0               0

       Former and Applied Values for Database Configuration
Description                                  Parameter   Current Value   Recommended Value
------------------------------------------------------------------------------------------
Default application heap (4KB)              (APPLHEAPSZ) = 256             256
Catalog cache size (4KB)              (CATALOGCACHE_SZ) = (MAXAPPLS*4)    260
Changed pages threshold                (CHNGPGS_THRESH) = 60              80
Database heap (4KB)                           (DBHEAP) = 1200            2791
Degree of parallelism                     (DFT_DEGREE) = 1               1
Default tablespace extentsize (pages)   (DFT_EXTENT_SZ) = 32              32
Default prefetch size (pages)         (DFT_PREFETCH_SZ) = AUTOMATIC       AUTOMATIC
Default query optimization class         (DFT_QUERYOPT) = 5               5
Max storage for lock list (4KB)              (LOCKLIST) = 100             AUTOMATIC
Log buffer size (4KB)                        (LOGBUFSZ) = 8               99
Log file size (4KB)                         (LOGFILSIZ) = 1000            1024
Number of primary log files                (LOGPRIMARY) = 3               8
Number of secondary log files               (LOGSECOND) = 2               3
Max number of active applications            (MAXAPPLS) = AUTOMATIC       AUTOMATIC
Percent. of lock lists per application       (MAXLOCKS) = 10              AUTOMATIC
Group commit count                          (MINCOMMIT) = 1               1
Number of asynchronous page cleaners   (NUM_IOCLEANERS) = 1               1
Number of I/O servers                   (NUM_IOSERVERS) = 3               4
Package cache size (4KB)                   (PCKCACHESZ) = (MAXAPPLS*8)    1533
Percent log file reclaimed before soft chckpt (SOFTMAX) = 100             320
Sort list heap (4KB)                         (SORTHEAP) = 256             AUTOMATIC
statement heap (4KB)                         (STMTHEAP) = 4096            4096
```

```
Statistics heap size (4KB)              (STAT_HEAP_SZ) = 4384           4384
Utilities heap size (4KB)               (UTIL_HEAP_SZ) = 5000           113661
Self tuning memory                    (SELF_TUNING_MEM) = ON            ON
    Automatic runstats                  (AUTO_RUNSTATS) = ON            ON
Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = 5000          AUTOMATIC
```

For details about how to generate recommendations for database configuration or tune configuration parameters with the DB2 configuration advisor, see "Configuration advisor" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.dbobj.doc/doc/c0052481.html*.

Best practices

Use the following design best practices for estimating system resources and capacity planning:

As database architects and administrators, you should constantly monitor system performance and revise capacity planning as the business and workload change.

- Run benchmarks on nonproduction systems to estimate resource consumption by workload and to plan the capacity needed.
- Use the DB2 configuration advisor and the self-tuning memory manager (STMM) to estimate requirements and tune the database. Run the workload after tuning the database. Then, collect the values for database memory configuration parameters. Roll them up to compute the total memory requirement for the database.
- Use DB2 Performance Expert or IBM InfoSphere Optim tools to monitor and collect resource consumption metrics as the workload is running.

For more details about how to use the Configuration Advisor, the Design Advisor, and the self-tuning memory manager, see "Best practices: Cost reduction strategies with DB2" at *http://www.ibm.com/developerworks/db2/bestpractices/reducingcosts/*.

Reliability, availability, and scalability

To keep up with today's increasingly global and competitive marketplace, your business enterprise architecture must have flexibility to grow with future strategic requirements and ensure business continuity throughout planned and unplanned outages.

For a mission critical 24x7 business enterprise, a single hour of downtime can translate to millions of dollars of lost revenue, not to mention the damage to a business reputation and the potential loss of customers. Global enterprises operate across time-zones and offer business services around the clock. Scheduled maintenance windows for system maintenance and upgrades no longer exist. Distributed enterprises need the ability to provide proximity of service in each geographic location, coupled with the ability to circumvent network failures or transmission times.

Reliability, availability, and scalability solutions

Some or all aspects of reliability, availability, and scalability (RAS) can be achieved by implementing the following solutions. They are listed in the order of their capability, from the least to best, in providing all three aspects.

Shared disk cluster
It provides high availability on node failure in the cluster. This solution provides only high availability and does not offer scalability, disaster recovery, or protection against disk corruption.

Disk mirroring technology
There are many solutions that provide commercial disk mirroring technology for implementing high availability or disaster recovery with shared disk cluster solution.

However, these solutions do not completely protect you against disk corruption. If the source disk is corrupted, the corrupted data is propagated to the target as well. Moreover, this solution does not offer instantaneous failover capability, which is critical for 24x7 business.

DB2 High Availability Disaster Recovery feature
It is a low-cost and easy to manage replication solution. It provides high availability and disaster recovery solution for both partial and complete site failures. It also provides instantaneous failover.

DB2 pureScale® feature
It is a shared disk architecture that allows business enterprise to transparently scale OLTP clusters dynamically on demand. It provides unlimited capacity, reliability, and continuous availability.

Partitioned database environments
A partitioned database environments is a shared-nothing architecture that allows the database manager to scale to hundreds of terabytes of data and hundred of CPUs across multiple database partitions to form a single, large database server.

These partitions can be located within a single server, across several physical machines, or a combination. The database data is distributed across multiple database partitions, offering tremendous scalability and workload parallelization across these partitions.

Typical OLTP workloads are short running transaction that access few random rows of a table. Partitioned database environments are better suited for data warehouse and business intelligence workloads due to the interinstance communication that occurs on each transaction.

The right RAS solution for your business

To determine the right RAS solution for your business, you must first define your high availability and disaster recovery objectives. To help define the objectives, analyze whether your current business enterprise has the infrastructure in place to provide RAS.

In order to identify the right solution, answer the following questions to understand the business impact when downtime occurs.

- What are the required and crucial aspects of RAS for your business?
- What measures are already in place to mitigate the risks of business downtime?
- When the business infrastructure is down due to planned or unplanned outages:
  - What are your business requirements and your service level agreements with the clients?
  - What is the impact to your business and your clients? Is it loss of revenue, reputation, future sales, present and potential clients?
  - What is an acceptable recovery window if disaster strikes?
  - How long would it take to bring the current infrastructure back online?

The following sections take a closer look at two of the DB2 solutions that provide all three aspects of RAS.

DB2 High Availability Disaster Recovery feature

DB2 High Availability Disaster Recovery (HADR) feature is an easy to use data replication feature that provides high availability and disaster recovery solution for both partial and complete site failures.

HADR replicates data changes from a source database (called the primary) to a target database (called the standby). Each database uses its own storage. The standby is kept in sync with the primary by perpetually applying the transaction logs received from the primary. On planned or unplanned outage, the standby can instantaneously fail over to service clients. The failover can be automated by using any cluster server software.

The failover is transparent to the clients, when combined with automatic client reroute (ACR) or a cluster server with virtual IP configuration. Tivoli System Automation for Multiplatforms (SA MP) software is the recommended cluster server for HADR failover automation, since it is tightly integrated through DB2 high availability (HA) interfaces.

The following diagram shows an example of an HADR environment:

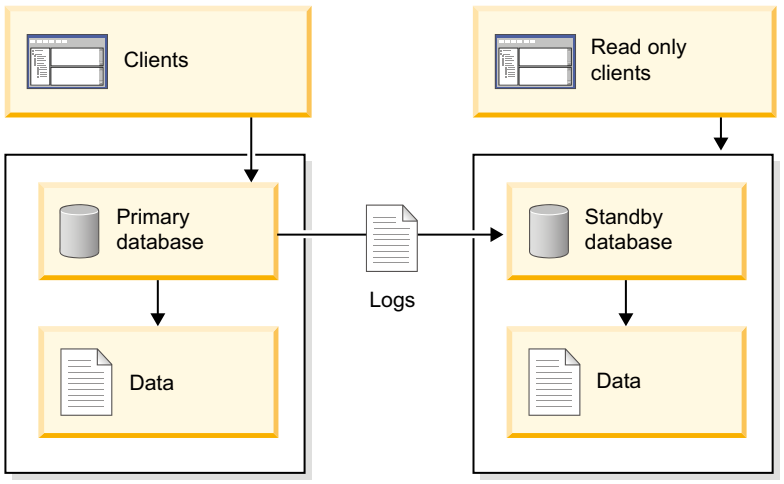

Figure 6. HADR environment

When HADR is right for your business

HADR is right for your business, when your business demands:

- A low-cost and simple solution that is easy to set up and administer for high availability and disaster recovery.
- Replication of the entire database.
- Instantaneous failover on planned and unplanned outages.
- Offload read-only workloads on standby to free up primary for business critical read or write workloads.
- High availability and disaster recovery solution for all platforms and on commodity hardware.
- Dynamic horizontal scaling is not a requirement.

For more details about recommendations on how to set up and maintain HADR, see "Best Practices: DB2 High Availability Disaster Recovery " at *http://www.ibm.com/developerworks/db2/bestpractices/hadr/*.

DB2 pureScale feature

The DB2 pureScale feature provides a shared-disk architecture that is used to transparently scale OLTP clusters without application changes while maintaining the highest availability levels available on distributed platforms.

It is primarily used to create active or active scale-out OLTP clusters. Members in a DB2 pureScale environment can be dynamically scaled out or scaled down according to business demand.

The DB2 pureScale feature reduces the risk and cost of business growth by providing unlimited capacity, reliability, continuous availability, and application transparency. The DB2 pureScale feature on IBM Power® Systems™ incorporates cluster caching facility (CF) technology on computers running UNIX operating systems such as x86 systems.

When the DB2 pureScale feature is right for your business

The DB2 pureScale feature is right for your business, when the business demands:

- Almost unlimited capacity and the ability to scale out or down dynamically without bring down the system.
- Continuous business availability and an "all active" architecture with inherent redundancy.
- Application transparency and load balancing.
- Reduced total cost of ownership.

The following diagram shows an example of a DB2 pureScale environment.

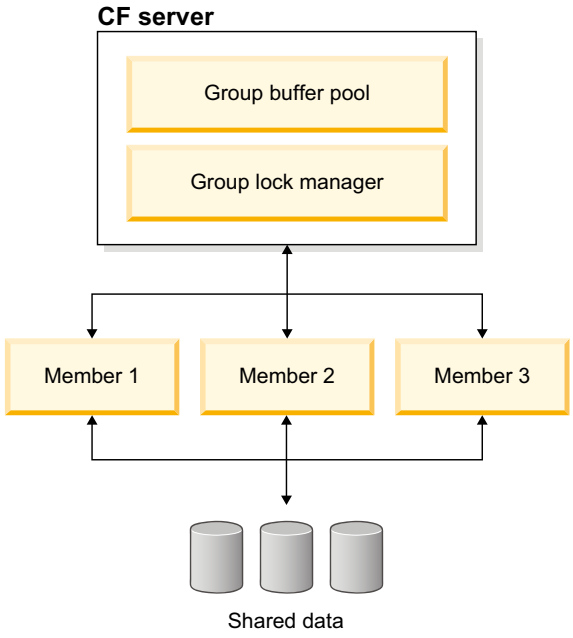

Figure 7. DB2 pureScale environment

A DB2 server that belongs to a DB2 pureScale environment is called a member. Each member has direct memory-based access to the centralized locking and caching services of the CF server. It can simultaneously access the same database for both read and write operations. Currently, the maximum number of members in a pureScale cluster is 128. The CF server provides centralized lock management services, a centralized global cache for data pages (known as the group buffer pool), and more.

Disaster recovery for DB2 pureScale deployment

DB2 Q Replication complements the capabilities of the DB2 pureScale feature by providing the following capabilities:

- Protection from disk and site failure by replicating the database to a remote site.
- Continuous availability during planned and unplanned outages.
- Active-active services on both source and target sites
- Offloading both read and write workloads to the target site, eliminating any possible contention with business critical workloads.

For more details about how to configure and deploy Q Replication in DB2 pureScale environments, see "DB2 best practices: Combining IBM DB2 pureScale with Q Replication for scalability and business continuity" at *http://www.ibm.com/developerworks/data/bestpractices/purescaleqreplication/index.html*.

Best practices

Use the following design best practices for RAS:

- Design your business infrastructure with a solid sizing and capacity planning for current and future needs as the business grows. Follow the sizing and capacity planning process described in this paper. For more details, see *"Database sizing and capacity management" on page 51*.
- Identify and eliminate single point of failures (SPOF) in the business infrastructure.
- Implement redundancy in your infrastructure such as networks and mirrored disks.
- Implement high availability and disaster recovery solutions in all layers of business infrastructure such as, database, application, and middleware.
- For DB2 databases:
  - Use separate high performing disks for data, transaction logs, and archived logs.
  - Use mirrored logs for redundancy.
  - Create a backup and restore plan for backing up databases, table paces, and transactional logs.
  - Use DB2 HADR as a simple solution for high availability and disaster recovery. For more details, see *"DB2 High Availability Disaster Recovery feature" on page 60*.
  - Use the DB2 pureScale feature with Q Replication to achieve scalability, reliability, continuous availability, and disaster recovery. For more details, see *"DB2 pureScale feature" on page 61*.

Operation and maintenance of your database systems

After a database system is put in production, the focus shifts towards ongoing maintenance of the database system. Day-to-day operational aspects that include performance management, problem diagnosis, and housekeeping are imperative to continue meeting the business service-level agreements.

The physical database design for your OLTP environment should include a schedule for operational and maintenance tasks. This section provides a summary of such activities.

Recovery strategy

As part of your overall RAS strategy, your recovery strategy plays an important role in meeting your RAS objectives. Regardless of the fact that there is redundancy at many levels, it is important to understand the business requirements when it comes to defining your recovery point objectives (RPOs) and your recovery time objectives (RTOs).

An RPO defines how much data loss is acceptable. An RTO defines the maximum amount of time to recover from a disaster such as disk failure, hardware failure, or operator error. The log files, the backup-image-retension period, and how many copies of log files and backup images to keep around are closely coupled with RTO and RPO.

For your business critical data, keep two or more copies of log files and backup images. For backup images, keep multiple generations of back images in addition to saving multiple copies of the same backup image. Use the multiple generations of backup images to perform point in time recovery before the last backup.

If an RTO is defined in the order of hours, restoring a database backup followed by a rollforward of the logs might be sufficient. However, if an RTO is defined in the order of seconds or minutes, you must use high availability software such as HADR.

Creating a recovery strategy is the beginning. After you devise a strategy, test it. Also do dry runs of your disaster recovery plans at regular intervals. The frequency depends upon the critical nature of the business application.

The database grows continuously over time. Any RPO or RTO objectives that were met during the last dry run might not be met today. Adjust physical design aspects such as faster I/O devices for recovery and spreading these devices across more spindles.

DB2 database products provide utilities that can help you design your recovery strategy. The utilities include online backups and snapshot backups.

Online backups can be performed while the database remains available. They are minimally intrusive and designed to concurrently run with other client activities.

Snapshot backup feature uses the fast copying technology of a storage device for an almost instant backup. This feature requires support from operating system or storage system.

For more details about database recovery, see "Data recovery" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.ha.doc/doc/c0052073.html*.

Maintenance window

For efficient running of database systems, maintenance operations including updating statistics, reorganizing tables and indexes, and performing backups must be scheduled on regular intervals.

DB2 supports automatic maintenance and online maintenance operations. You can continue to access the database while the maintenance operation is carried out when running an online maintenance operation.

If you choose a maintenance window when workload is light, online maintenance operations use throttling to use free resources and boost performance. Also, it reduces the possibility of conflict with regular application processing because maintenance operations acquire certain locks that could cause applications to wait.

Performance monitoring and tuning

As the database size and characteristics changes, you can address your business requirements proactively by regularly monitoring and tuning your databases before problems occur.

DB2 database products offer many tools such as the **db2pd** command, system-defined administrative views and routines, and the IBM InfoSphere Optim Performance Manager. The **db2pd** command is a popular, non-intrusive, command-line tool that you can use to monitor performance. The system-defined administrative views and routines provide an easy-to-use application programming interface through SQL. The IBM InfoSphere Optim Performance Manager is a web console that you can use to isolate and analyze typical database performance problems.

As a database administrator, you can carry out any necessary actions based on matrices reported by the monitoring tools. If your response time and throughput business objectives are not met, your possible actions might include adding more memory, increasing size of buffer pool, moving a table into its own table space with associated buffer pool, creating new indexes, or creating materialized views.

Testing environments

A change in a database requires functional, performance, and stability testing before it can be rolled out in a production environment.

Typical changes are the occasional adoption of new DB2 features or database tuning. Directly changing a production environment can be risky. The changes might affect the availability of a business system.

You can create a test environment with representative data and workload to test these changes instead of using your production environment. The test environment does not have to have same size as the production system. It can be a smaller subset of the production environment with smaller data set. DB2 offers many tools to create a parallel test system, including the IBM InfoSphere Optim tools.

Best practices

Use the following design best practices for operation and maintenance:

- Design a recovery strategy for your databases and test it. For more details, see *"Recovery strategy" on page 65*.
- Configure the **logarchmeth1** and **logarchmeth2** database configuration parameters to archive log files to multiple locations. For more details about configuring database logging, see *"Database transaction logs" on page 35*.
- Choose a maintenance window when workload is light to run online administration commands. For more details, see *"Maintenance window" on page 66*.
- Use a test environment to experiment with changes to adopt new DB2 features or to tune the database. For more details, see *"Testing environments" on page 66*.

Best practices summary

The following list summarizes the most relevant best practices for physical database design in OLTP environments.

Data modeling

Use InfoSphere Data Architect to perform data modeling and database physical design tasks such as:

- Create a logical data model and then transform it into a physical data model. Work with the physical data model to plan the physical storage for table spaces, indexes, or views by adding storage objects.
- Generate DDL scripts that will help you to deploy the DB2 database. Run these script to create the database and its objects on DB2 server.
- Revise your physical data model as your business needs change and make changes to the data model accordingly.

For more details, see *"Data modeling" on page 9*.

Designing storage systems

- Use storage server level hardware that has RAID array capability. RAID5 offers a balance between cost, redundancy, and performance.
- Disable storage-server level read-ahead since OLTP workloads do not exhibit sequential I/O and do not benefit from read-ahead.
- Use RAID 1+0 or RAID5 as log devices for better performance and higher RAS.
- If the storage system has a battery backup, enable write-behind.
- If the hardware level RAID support is not available, use logical volume manager level RAID.
- Use as many hardware level RAS features and performance capabilities as possible. For example, hardware RAID features tend to be faster than software RAID features in the operating system or volume manager level.

For more details, see *"Storage systems" on page 11*.

Designing table spaces

- Prefer automatic storage to DMS table spaces. Automatic storage offers an important advantage with automatic container management.
- Use CIO or DIO in table spaces to bypass file system buffers and prevent double buffering, especially in databases that you migrated from Version 9.1. Ensure that the buffer pools are tuned appropriately. The result is better I/O performance. For details, see *"Table space design for OLTP workloads" on page 13*.
- Using table spaces with 8 KB or 16 KB page sizes can let you store more data on disks with lesser impact on I/O and buffer pool storage costs than 32 KB page size. If you use a larger page size and access is random, you might need to increase the size of the buffer pool to achieve the same buffer pool hit ratio for reading that you had with the smaller page size. For details, see *"Table space page sizes" on page 14*.

For more details, see *"Table spaces and Buffer pools" on page 13*.

Designing buffer pools

- Create additional buffer pools for each page size used in table spaces. Having more than one buffer pool allows you to configure the memory used by the database to improve overall performance. Care must be taken in configuring additional buffer pools.
- Explicitly set the size of buffer pools or enable the STMM to tune buffer pool sizes automatically. For details, see *"Buffer pool design" on page 15*.
- Associate different buffer pools for temporary table spaces and permanent table spaces for data and large objects to avoid possible buffer pool contention. For details, see *"Buffer pool design" on page 15*.
- Set the **num_iocleaners** parameter to `Automatic` and the **DB2_USE_ALTERNATE_PAGE_CLEANING** registry variable to `YES`. For details, see *"Page cleaning activity" on page 16*.
- Monitor buffer pool usage by using the **db2pd -bufferpools** command.

For more details, see *"Table spaces and Buffer pools" on page 13*.

Selecting data types

- Always try to use a numeric data type over a character data type, taking the following considerations into account:
  - When creating a column that holds a Boolean value ("YES" or "NO"), use a DECIMAL(1,0) or similar data type. Use 0 and 1 as values for the column rather than "N" or "Y".
  - Use integers to represent codes.
  - If there are less than 10 code values for a column, the DECIMAL(1,0) data type is appropriate. If there are more than 9 code values to be stored in a column, use SMALLINT.
- Store the data definitions as a domain value in a data modeling tool, such as InfoSphere Data Architect, where the values can be published by using metadata reporting.
- Store the definition of the values in a table in a database, where the definitions can be joined to the value to provide context, such as "text name" or "description".

For more details, see *"Data types" on page 19*.

Designing tables

- Use range-clustered tables for tightly clustered data to provide fast direct access to data.
- Use table partitioning to logically organize tables, improve recovery efficiency, and improve data roll-out efficiency.
- Use MDC tables to organize and cluster data based on multiple dimensions and guarantee automatic clustering.
- Partition tables with large-scale data by both range partitioning and multidimensional clustering to take advantage of data partitions and block elimination to improve query performance.
- Use the DB2 design advisor to get recommendations on the repartitioning of tables, the conversion to multidimensional clustering (MDC) tables, and the deletion of unused objects. For more details, see "The Design Advisor" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.perf.doc/doc/c0005144.html*.

For more details, see *"Tables" on page 23*.

Designing indexes

- Use an index only where there is a clear advantage for frequent access exists. For details, see *"Index guidelines for OLTP workload" on page 29*.
- Use columns that best match with the most frequent used queries as index keys. For details, see *"Index guidelines for OLTP workload" on page 29*.
- Use include columns for two or more columns that are frequently accessed together to enable index only access for queries. For more details, see *"Index guidelines for OLTP workload" on page 29*.
- Use partitioned indexes for partitioned tables. For more details, see *"Indexes for range partitioned tables" on page 31*.
- Create clustered index on columns that have range predicates. Indicate a PCTFREE value to reduce the need of index reorganization. For OLTP workloads with significant insert or update operations, use a large PCTFREE value. For more details, see *"Clustering indexes" on page 31*.
- Create indexes over XML columns for faster performance on queries over XML data. For more details, see *"Indexes for tables with XML data" on page 32*.
- Use the Design Advisor to get recommendations for RID-based indexes on MDC tables or MDC dimensions for a table. Choosing the right dimensions for your OLTP workload is important because block indexes are automatically created for each dimension. For more details, see *"Indexes for tables with XML data" on page 32*.
- Eliminate indexes that are never accessed. Use Design Advisor to determine whether you have indexes that have not been accessed. For more details, see *"Adjust indexes design" on page 32*.
- Avoid redundant indexes. Use Design Advisor to determine whether you have redundant indexes. For more details, see *"Adjust indexes design" on page 32*.

For more details, see *"Indexes" on page 29*.

Database logging

- Use archive logging in production environments to be able to perform many recovery operations including, online backup, incremental backup, online restore, point-in-time rollforward, and issuing the RECOVER DATABASE command.
- Consider enabling the **trackmod** database configuration parameter for incremental backups to track database modifications so that the **BACKUP DATABASE** command can determine which subsets of database pages should be included in the backup image for either database backups or table space backups.
- Use mirror log path to improve high availability. For more details, see *"Mirror log path" on page 36*.
- Configure secondary log files to provide additional log space on a temporary basis.
- Use data and index compression to improve performance of administrations tasks such as database and table space backups because the backups on compressed data take less time.
- Consider the I/O adapter or bus bandwidth requirements for transaction logging. If requirements are not adequate, I/O usage might result in a bottleneck. If high availability is a concern, look into operating system level support for I/O multi-pathing.

For more details, see *"Database transaction logs" on page 35*.

Data and index compression

- For row compression:
  - Identify tables that are good candidates for row compression. Understanding typical SQL activity against the table can help you determine whether it is a good candidate for row compression. For more details, see *"Good candidates for row compression" on page 40*.
  - Determine which tables to compress by using compression ratio estimates. For more details, see *"Estimates for compression ratios" on page 40*.
  - Use the INSPECT command or the ADC to build a compression dictionary instead of reorganizing the table. For more details, see *"Building a compression dictionary" on page 41*.
  - If you are reorganizing a table to compress it, do not specify an index to cluster the data. Eliminating the extra processing for clustering minimizes the time it takes to perform the compression. For more details, see *"Clustering data during table reorganization" on page 42*.
  - If you are using row compression and storage optimization is of higher priority than the extra time it takes to perform a compressed backup, use backup compression. For more details, see "Backup compression" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.ha.doc/doc/c0056483.html*.
- Keep index compression enabled. For more details, see *"Index compression" on page 43*.
- Use value compression in table with many column values equal to the system default values or NULL. Value compression can be combined with other compression methods. For more details, see "Value compression" at *http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.dbobj.doc/doc/c0056482.html*

For more details, see *"Data and index compression" on page 39*.

Query design

- For single queries:
  - **Use index scans rather than table scans.** If a table scan was used, try reorganizing the table and index. After recompiling the query, determine whether the access plan uses an index scan with the explain facility.
  - **Avoid complex expressions in search conditions.** Such expressions can prevent the optimizer from using catalog statistics to estimate an accurate selectivity. Also, they might limit the choices of access plans that can be used to apply the predicate.
  - **Use join predicates on expressions to promote the use of nested loop join method.** Joins that involve only a few rows, such as OLTP queries, typically run faster with nested-loop joins.
  - **Create indexes on columns that are used to join tables in a query.** Indexes on these columns can speed up local predicates.
  - **Avoid data type mismatches on join columns.** In some cases, data type mismatches prevent the use of hash joins.
  - **Do not use no-op expressions in predicates to change the optimizer estimate.** A "no-op" expression of the form `COALESCE(X, X) = X` introduces an estimation error into the planning of any query.

- **Avoid non-equality join predicates.** Avoid these predicates because the join method is limited to nested loop.

- For multiple concurrent queries:
  - **Use the minimum isolation level that satisfies your application needs.** The less restrictive isolation level, the fewer locks required, and the less memory and CPU resources are consumed.
  - **Conditions that might cause lock escalations that should be avoided.** Lock escalations reduce concurrency and consume system resources.
- Prevent application deadlocks by:
  - Closing cursors to release the locks that they hold.
  - Closing a CURSOR WITH HOLD before issuing a COMMIT statement.
  - Using the LOCK TABLE statement appropriately.

  For more details, see *"Application deadlocks" on page 46*.
- If your application uses complex SQL requests, use DB2 parallelism on symmetric multiprocessor (SMP) computers. In general, OLTP environments do not benefit from enabling DB2 parallelism on SMP computers.
- Maintain current statistics for tables and indexes. Update statistics regularly on any critical tables, including system catalog tables. For more details, see *"Catalog statistics" on page 47*.
- Use the explain facility to monitor and evaluate query performance changes.
- Use event monitors to monitor deadlocks, locking, and units of work. For more details, see *"DB2 event monitors" on page 48*.
- Use a low optimization class such as 0 or 1 for queries with a run time of less than one second. Use a higher optimization class such as 3, 5, or 7 for longer running queries that take more than 30 seconds. For more details, see *"Optimization classes" on page 48*.

For more details, see *"Query design" on page 45*.

Database sizing and capacity management

As database architects and administrators, you should constantly monitor system performance and revise capacity planning as the business and workload change.

- Run benchmarks on nonproduction systems to estimate resource consumption by workload and to plan the capacity needed.
- Use the DB2 configuration advisor and the self-tuning memory manager (STMM) to estimate requirements and tune the database. Run the workload after tuning the database. Then, collect the values for database memory configuration parameters. Roll them up to compute the total memory requirement for the database.
- Use DB2 Performance Expert or IBM InfoSphere Optim tools to monitor and collect resource consumption metrics as the workload is running.

For more details, see *"Database sizing and capacity management" on page 51*.

Reliability, Availability, and Scalability

- Design your business infrastructure with a solid sizing and capacity planning for current and future needs as the business grows. Follow the

sizing and capacity planning process described in this paper. For more details, see *"Database sizing and capacity management" on page 51*.

- Identify and eliminate single point of failures (SPOF) in the business infrastructure.
- Implement redundancy in your infrastructure such as networks and mirrored disks.
- Implement high availability and disaster recovery solutions in all layers of business infrastructure such as, database, application, and middleware.
- For DB2 databases:
  - Use separate high performing disks for data, transaction logs, and archived logs.
  - Use mirrored logs for redundancy.
  - Create a backup and restore plan for backing up databases, table paces, and transactional logs.
  - Use DB2 HADR as a simple solution for high availability and disaster recovery. For more details, see *"DB2 High Availability Disaster Recovery feature" on page 60*.
  - Use the DB2 pureScale feature with Q Replication to achieve scalability, reliability, continuous availability, and disaster recovery. For more details, see *"DB2 pureScale feature" on page 61*.

For more details, see *"Reliability, availability, and scalability" on page 59*.

Operation and maintenance

- Design a recovery strategy for your databases and test it. For more details, see *"Recovery strategy" on page 65*.
- Configure the **logarchmeth1** and **logarchmeth2** database configuration parameters to archive log files to multiple locations. For more details about configuring database logging, see *"Database transaction logs" on page 35*.
- Choose a maintenance window when workload is light to run online administration commands. For more details, see *"Maintenance window" on page 66*.
- Use a test environment to experiment with changes to adopt new DB2 features or to tune the database. For more details, see *"Testing environments" on page 66*.

For more details, see *"Operation and maintenance of your database systems" on page 65*.

Conclusion

Physical database design is the single most important aspect of any database. It affects the scalability, efficiency, maintainability, and extensibility of a database much more than other aspects of database administration.

Although database design can be complex, a good design improves performance and reduces operational risk. Mastering good physical database design is a cornerstone of professional database administrators.

Important references

These important references provide further reading about physical database design and related aspects.

- Best practices: Cost reduction strategies with DB2 at *http://www.ibm.com/developerworks/db2/bestpractices/reducingcosts/*
- Best Practices: Database storage at *http://www.ibm.com/developerworks/db2/bestpractices/databasestorage/*
- Best Practices: Deep Compression at *http://www.ibm.com/developerworks/data/bestpractices/deepcompression/*
- Best Practices: DB2 High Availability Disaster Recovery at *http://www.ibm.com/developerworks/db2/bestpractices/hadr/*
- DB2 best practices: Combining IBM DB2 pureScale with Q Replication for scalability and business continuity at *http://www.ibm.com/developerworks/data/bestpractices/purescaleqreplication/index.html*
- Demoted I/O requests may lead to DB2 performance problems at *https://www-304.ibm.com/support/docview.wss?uid=swg21469603*
- Getting started with IBM InfoSphere Data Architect at *http://public.dhe.ibm.com/software/dw/db2/express-c/wiki/Getting_Started_with_IDA.pdf*.
- 15 best practices for pureXML® performance in DB2 at *http://www.ibm.com/developerworks/data/library/techarticle/dm-0610nicola/*
- DB2 Best Practices at *http://www.ibm.com/developerworks/db2/bestpractices/*
- DB2 database product documentation at *https://www-304.ibm.com/support/docview.wss?rs=71&uid=swg27009474*
- Database Fundamentals at *http://www.ibm.com/developerworks/wikis/display/db2oncampus/FREE+ebook+-+Database+fundamentals*
- Lightstone, et al., *Physical Database Design: the database professional's guide to exploiting indexes, views, storage, and more*, ISBN 0123693896S. Morgan Kaufmann Press, 2007.
- Lightstone, et al., *Database Modeling & Design: Logical Design*. ISBN 0126853525T, 4th ed. Morgan Kaufmann Press, 2005.

Contributors

Contributors that provided technical information used by the authors.

Serge Boivin
Senior Writer

DB2 Information Development

Garrett Fitzsimons
Data Warehouse Lab Consultant

Warehousing Best Practices

Gary Jin
Client Technical Professional

Information Management

The authors of the "Best Practices Physical Database Design" paper contributed by allowing reuse of the paper content:

Agatha Colangelo
DB2 Information Development

Sam Lightstone
Program Director and Senior Technical Staff Member

Information Management Software

Christopher Tsounis
Executive IT Specialist Information Management

Technical Sales

Steven Tsounis
IT Specialist

Information Management Technical Sales

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.

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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. 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.

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.

Index

O

P

Q

R

S

T

W

X

Printed in USA