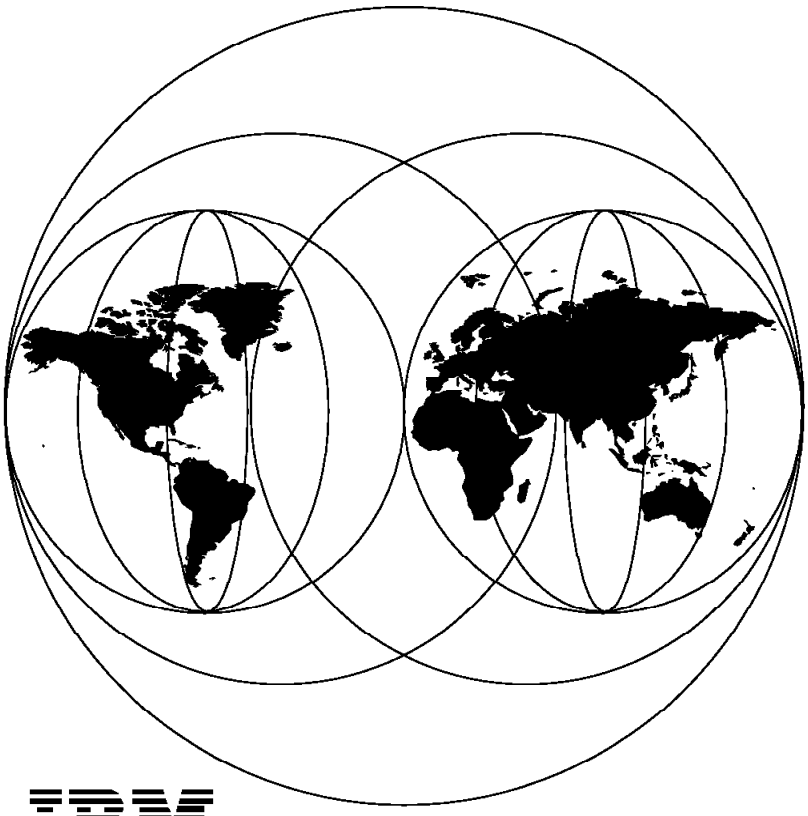
SG24-2514-01

DB2 Parallel Edition for AIX: Concepts and Facilities

December 1996



IBM

International Technical Support Organization

Austin Center

International Technical Support Organization

SG24-2514-01

DB2 Parallel Edition for AIX: Concepts and Facilities

December 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix A, “Special Notices” on page 255.

Second Edition (December 1996)

This edition applies to DB2 Parallel Edition for AIX Version 1.2 of DB2 Parallel Edition for AIX, Program Number 5765-328, for use with the AIX operating system.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Building 045 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

DB2 Parallel Edition for AIX is one of IBM′s products for high-demand database uses, such as data warehousing and decision support. It is a full-function relational database management system that parallelizes SQL operations.

This redbook includes all the features and functions that are found in Version 1.2 of the product. Database administrators and users from the novice to the expert will find this book to be a valuable resource. Examples are given throughout to illustrate the easy-to-use features and performance enhancements found in Version 1.2.

How This Redbook Is Organized

This redbook contains 272 pages. It is organized as follows:

- Chapter 1, “Overview”

  This chapter provides an introduction to the parallel database environment. Concepts discussed are parallel architecture, parallel processing and parallel process flow.

- Chapter 2, “Hardware Configurations”

  This chapter provides an overview of some of the possible hardware on which DB2 PE can be installed and used. High Availability Cluster Multi-Processing (HACMP) is also discussed.

- Chapter 3, “Concepts and Data Placement”

  This chapter describes parallel database concepts as they relate to DB2 Parallel Edition for AIX. The concepts that are covered are instance, nodegroup, data partitioning, and the partitioning map.

- Chapter 4, “Parallel Processing”

  This chapter discusses the parallel processing for all types of SQL operations. Covered in this chapter are SQL calls, operations and optimization, buffered inserts, join strategies, table queues, and locking.

- Chapter 5, “Parallel Utilities”

  Some of the utilities found in DB2 PE are discussed here, such as data loading, redistribution, backup and restore, and recovery.

- Chapter 6, “Installation and Configuration”

  This chapter looks at the installation and configuration of the DB2 PE product, including HACMP configurations.

DB2 Parallel Edition V 1.2 Enhancements

DB2 Parallel Edition Version 1.2 adds additional functions and features. Some of these enhancements are:

Performance

Parallel Optimizer

- Additional optimization techniques are employed by the Parallel Edition cost-based optimizer to reduce the elapsed time of complex queries. In addition, system catalog concurrency has been increased by improved lock management techniques.

Not Initially Logged Tables

- Logging can be disabled for SQL operations occurring against a newly created table. This feature is especially useful for customers who populate tables by selecting from existing tables.

Function

DRDA Application Server

- DB2 Parallel Edition now allows host applications, such as Query Management Facility (QMF), access to DB2 Parallel Edition. A section about Distributed Relational Database Architecture (DRDA) Application Server including SNA Server/6000 configuration can be found in 6.5, “DRDA Application Server Feature” on page 246.

Outer Join

- An Outer Join operation allows unmatched rows of a join operation to be returned to the client application. This is especially useful in data warehouse applications. For example, a banking application using an outer join can create a report that shows customers that don’t have accounts or accounts that are not related to any customers. In this example, the use of outer join allows for data consistency problems to be highlighted.

CASE expression and DIGITS scalar function

- The CASE expression allows a result expression to be selected on the evaluation of one or more search conditions. This reduces coding effort for your 3GL applications, and improves performance by reducing SQL calls. The DIGITS scalar function can be used to return a character-string representation of a number.

Join operations, including examples of outer joins, the CASE expression, and the DIGITS scalar function can be found in Chapter 4, “Parallel Processing” on page 71.

Systems Management Enhancements

Database Director

- The Database Director delivers an graphical user interface to manage the DB2 Parallel Edition environment. It serves as a control center for database system administration and simplifies administration by providing a single-system image of the parallel environment. An example of using the Database Director can be found in 5.15, “DB2 Parallel Edition Database Director” on page 216.

Query Governor

- Using rules set by the database administrator, the DB2 Parallel Edition Query Governor can control resource usage of individual users and applications. This is accomplished by adjusting the application′s priority level. Resource usage can be controlled by factors such as CPU seconds, rows selected, elapsed time and locks held. For more detail on the Query Governor, see 5.13, “Governor Utility” on page 203.

Ease of Use Enhancements

AutoLoader Utility

- The AutoLoader Utility is a sample tool that can transfer data from one system to the AIX system, partition that data and finally load the data on the corresponding nodes. It consists of a set of shell scripts. Autoloader uses pipes between the ftp, db2split, and db2load utilities, removing the need for having temporary space for your load files. Detailed information about the Autoloader Utility, including examples of usage can be found in 5.4, “Autoloader Utility” on page 164.

db2batch

- A benchmark tool (db2batch) is provided. This tool reads SQL statements from either a flat file or standard input, dynamically describes and prepares the statements, and returns an answer set. It also provides the added flexibility of allowing you to control the size of the answer set, as well as the number of rows that should be sent from this answer set to an output device. Examples using the db2batch utility can be found in 5.14, “db2batch Tool” on page 212.

The Team That Wrote This Redbook

This second edition was produced by a team of specialists from around the world working at the International Technical Support Organization Austin Center.

IBM USA:
Gus Branish
Quan Cung
Dan Gibson
Randy Holmes

IBM France
Jean-Christophe Brun

IBM Germany
Michael Mueller

The advisor for the first edition of this redbook was:

Calene Janacek
International Technical Support Organization, Austin Center

The authors of the first edition of this redbook were:

IBM Taiwan
Annie Pan

IBM Italy
Giovanni Punzo

IBM UK
Kevin Robson

IBM Japan
Kayoko Sugahara

Thanks to the following people for their invaluable contributions to this project:

IBM Toronto Laboratory:
George Chan
Ken Chen
Chris Eator
Alex Lui
John Lumby
Barbara Wong

IBM Toronto
Frankie Wong

IBM Education
Melanie Stopfer

IBM ITSO, San Jose Center:
Silvio Podcameni

ITSO Austin Center Editors
Rebeca Rodriguez
Marcus Brewer

A special thanks is extended to the management and team members of the SP Benchmark Center in Poughkeepsie for their superb support. In particular:

IBM USA
Joyce Mak

Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!

Chapter 1. Overview

This book is intended to be a general introduction to the parallel database world and to IBM′s parallel database offering for AIX, DB2 Parallel Edition for AIX/6000, (referred to as DB2 Parallel Edition). In this chapter, we will review the concepts behind parallel databases and the reasons for pursuing this type of database. The chapter then discusses some of the advantages and disadvantages of the different types of parallel architected solutions that exist today and the architecture that DB2 Parallel Edition has chosen as its model.

The chapter is organized as follows:

- Why parallel databases were developed
- Concepts and definitions used in the parallel database architecture and processing

1.1 The Need for Parallelism

The development of parallel database systems has come from an increase in the use of Relational Database Management Systems (RDBMS). There is also the expectation that responses should be received faster than ever before. Users may want their queries answered either by directly entering input from a terminal screen or through an application program. High-level query languages, such as SQL (Structured Query Language), were developed to access an RDBMS. Queries are structured to generate complex reports, searching gigabytes or more of data. These queries have grown not only in the amount of data searched, but also in complexity.

Based on the current rate of increase in the complexity of queries and volumes of data, it is not expected that the processing capacity of any single processor, or even a closely-coupled multiprocessor, will be sufficient to provide acceptable response times. It may be more cost effective to have more machines with less individual CPU power accessing data than to have one large CPU. Consider the example of scanning a table in a serial RDBMS. At present, a table is scanned at a rate of 2.25 MB/second. If the table size is 10 GB, it would take one processor over an hour to scan a table. If that processing could be distributed evenly among 128 processors, the time required to scan that same table would drop to 35 seconds. There are also architectural limits in terms of the number of disks that can be attached to a single processor. This limits the size of a database held on a single processor.

The amount of data and access time must also be considered. Market studies have been collected that show a need for companies to be able to store terabytes of data in the next two years. Companies want to maintain more customer records, with more detailed information about customers, for a longer period of time. For example, one of the television video-rental companies maintains a database of more than 36 million records with daily transactions of over two million. Another example is that of new database structures that involve large amounts of storage, such as multimedia data types.

The other factor to support the argument for multiprocessors is increased computational power. Decision support algorithms are getting more sophisticated, and more complex queries are being done over databases. The

database has become a potent new tool for selling. An example of this usage is in the area of target marketing. Instead of doing broadcast advertising, a company will perform more sophisticated selections and will send mailings to only a subset of, hopefully, more interested customers. This results in enormous savings for the company and less junk mail for the customer. In order to do this type of processing, more complex queries must be issued against the database, and much more information about the customer must be maintained.

An RDBMS can provide two types of parallel support in terms of multiprocessing systems:

- Parallel Transactions
- Parallel Queries

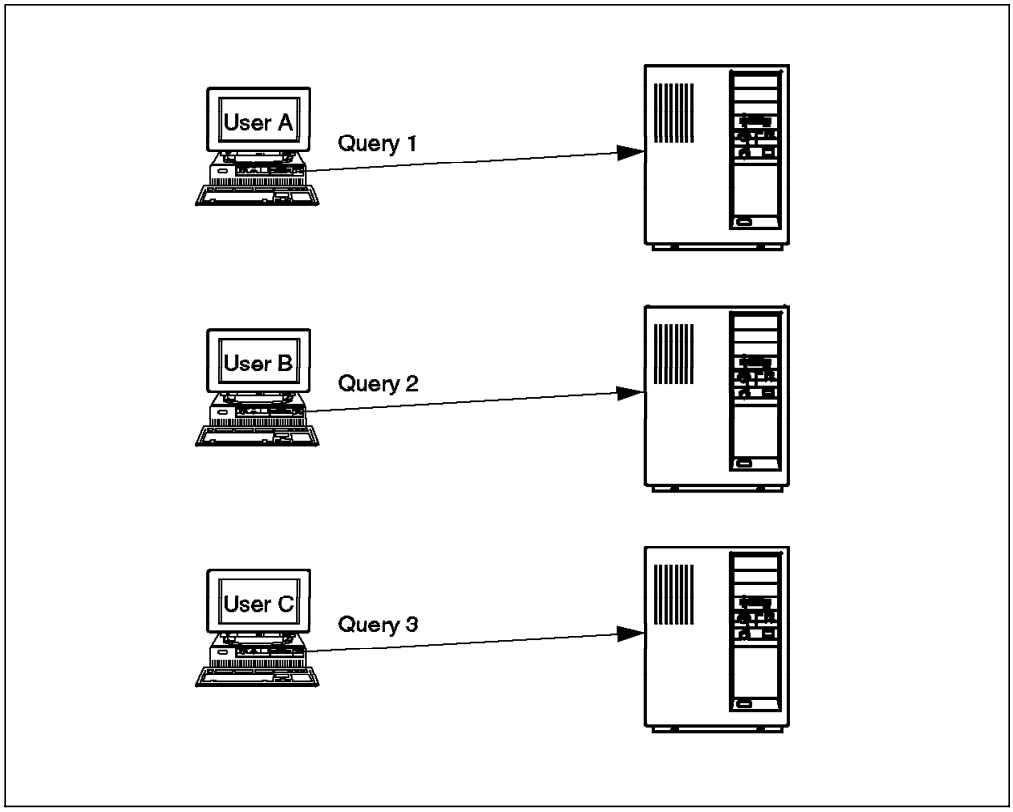


Figure 1. Parallel Transactions

Figure 1 shows a parallel transaction. Multiple queries are processed in parallel. Each query is on a different CPU.

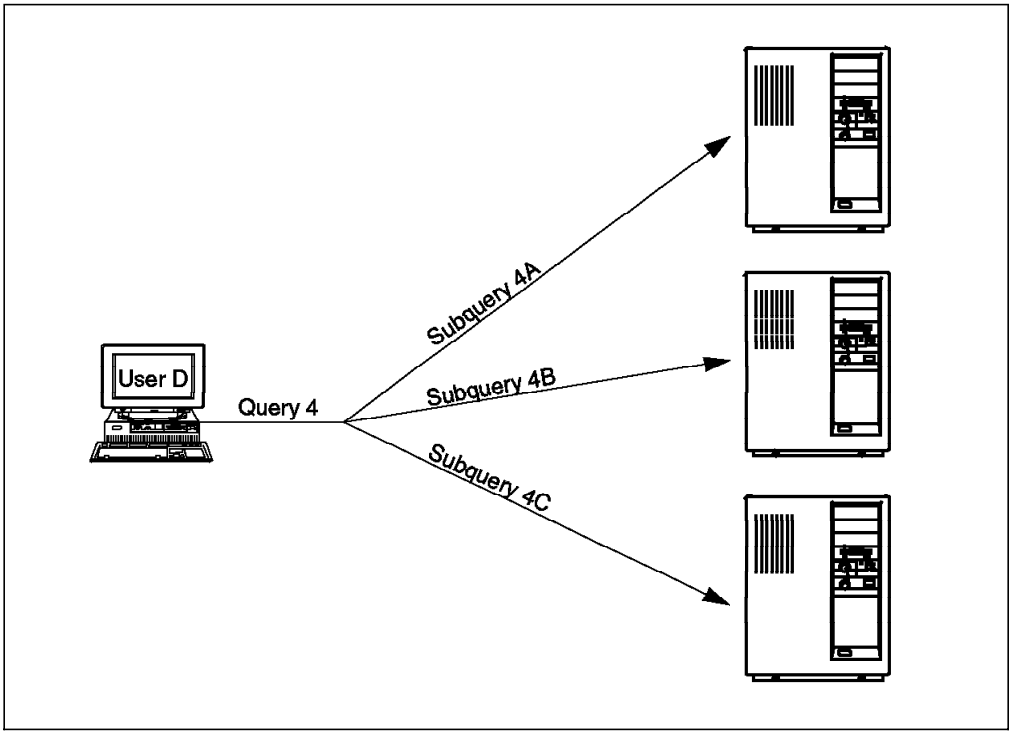


Figure 2. Parallel Query

Figure 2 shows a parallel query being broken into smaller steps. Each step, or subquery, can be executed on a different processor. These subqueries are executed in parallel.

An RDBMS is well suited for designing parallel queries. The result of a query can also be used by a further query with another operator. In this way, relational operators, as well as the input to and output from the results of operations, can be parallelized.

There are many benefits that can be obtained from using a parallel database implementation:

- Increased transaction throughput
- Better price/performance
- Reduced processing time for queries
- Flexible scalability
- Ability to store more data

1.2 Concepts and Definitions in Parallel Database Architecture

This section will cover the following areas with respect to parallel database architecture:

- Different types of parallel architecture
- Process flow

Parallel architecture involves hardware and system software and their influence over an RDBMS. Process flow is how the RDBMS handles operations. The process flow is closely linked to the architectural implementation.

1.2.1 Parallel Architecture

There are three hardware elements that characterize parallel architecture:

- Memory
- Disk
- Network types

A parallel database running on a machine uses memory and disk and communicates to other machines on a network. There are three different hardware architectures for a parallel database:

- Shared Nothing
- Shared Disk
- Shared Memory

1.2.1.1 Shared Nothing

Shared nothing architecture is realized when loosely coupled processors are linked by some high-speed interconnection. Each processor has its own memory and accesses its own disks. This can be seen in Figure 3. The network is one that supports Internet Protocol (IP) connections. There are performance differences between network types, and these are discussed in Chapter 2, “Hardware Configurations” on page 15.

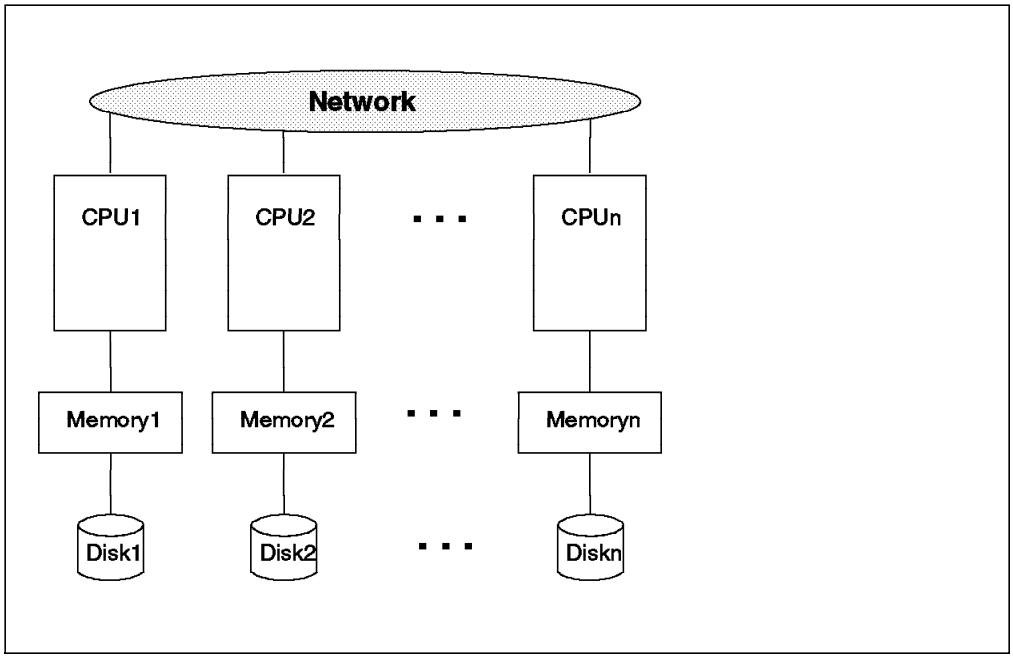


Figure 3. Shared Nothing Architecture

Examples of machines that implement this architecture are:

- IBM RISC System/6000 Scalable Power Parallel Systems (RS/6000 SP)
- Clusters of workstations
- nCube
- Teradata

The advantages of this type of architecture are the following:

- Scalability in terms of database size and number of processors
- Performance gains from not sharing resources across a network

Total memory is a fixed capacity. By increasing the number of machines, you can exceed that fixed amount because the memory is shared among machines. The same is true for total disk capacity. The other advantage that could be gained is in the number of operations that are performed. Each machine only needs to do part of the work.

This is the best suited architecture for parallel queries. The query is divided among processors. The advantages are:

- Processing is more distributed
- The database can manage a larger amount of data

The performance gains can be almost linear if an even distribution of data is obtained. For more information on data distribution, refer to Chapter 3, “Concepts and Data Placement” on page 35.

Performance gains are assisted by the concept of function shipping (see 1.2.3.1, “Function Shipping” on page 10). Function shipping assists in the reduction of network traffic since functions are shipped instead of data.

While machines, such as the RS/6000 SP, have their own fast communication network, clusters of RS/6000s may need to be interconnected by a high-speed network to achieve good performance when processing parallel operations.

The disadvantages of this architectures are:

- Dependence on the optimizer for the access strategy to data
- Load balancing
- Availability
- Homogeneous environment

If the design of the optimizer is good, this does not have to be a disadvantage.

The even distribution of data is another factor that can affect performance. DB2 Parallel Edition provides utilities that allow the database administrator to redistribute the data among machines.

Availability may be a concern in the parallel database environment. With “n” machines, a failure of one of them is “n” times more likely to occur.

The last consideration is one of similarity in environment. It is possible to have different processor types. For maximum performance from the optimizer, a homogeneous environment with a single network and similar processors is desired.

1.2.1.2 Shared Disk

With shared disk architecture, several distributed memory processors are capable of accessing all the same data stored on disk. This means that every processor has its own memory, but it has a global view of all the data. This can be implemented either by hardware or software. Figure 4 on page 6 is an example of shared disk architecture.

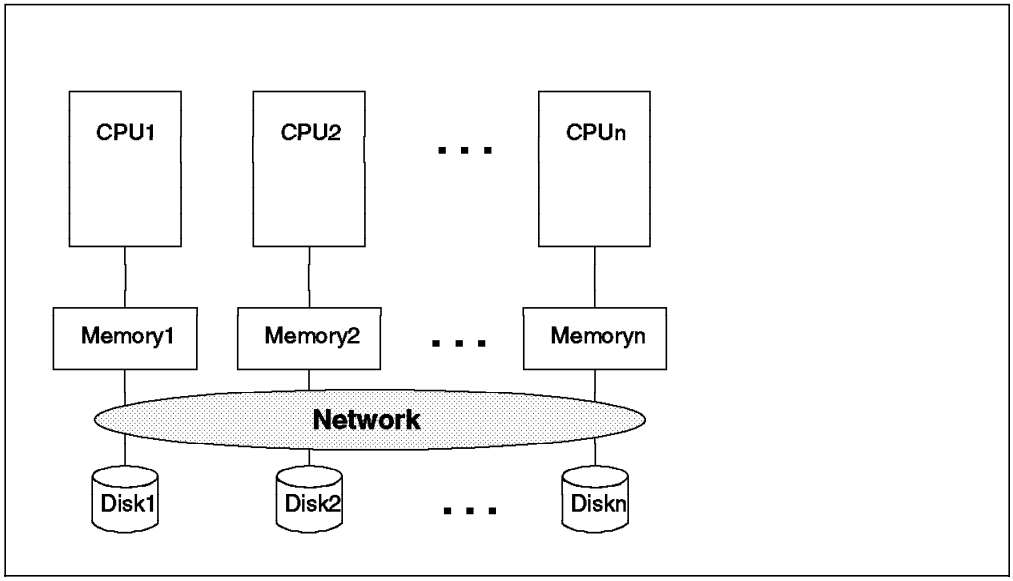


Figure 4. Shared Disk Architecture

Examples of machines that implement this architecture are:

- Clusters of RISC/6000s using HACMP Concurrent Logical Volume Manager
- RS/6000 SPs using Virtual Shared Disk (VSD)
- Digital Equipment Corporation (DEC) Virtual Address eXtension (VAX) mini-computers

The natural architecture for an RS/6000 SP is shared nothing, but using a component of the system management software called Virtual Shared Disk (VSD), it is possible to simulate shared disk behavior. For further information about VSD, refer to the *SP2 Administration Guide* .

Possible advantages of this architecture are:

- Availability
- A heterogeneous environment

A possible advantage of this architecture is with an HACMP Concurrent Logical Volume Manager environment. So, even if one machine fails, access to the database will not be affected since other machines are still able to access that data.

Using the VSD architecture allows you to operate in a heterogeneous environment. For example, you could designate your machines dedicated to parallel database in two categories:

- Data storage machines
- Processing machines

The disadvantages of this architecture are:

- Limited scalability
- Increased data traffic across the network
- Data integrity

With shared disk architecture, scalability is limited. Since the storage is external, a global data-sharing mechanism must also be provided. There are two possibilities to consider. If global data sharing is realized by hardware, then only limited scalability is possible due to hardware constraints. For example, the current implementation of the HACMP Concurrent Logical Volume Manager allows eight nodes to share IBM 7133 SSA Disk Subsystems, but only four nodes can share IBM 7135 RAIDiant Array subsystems due to hardware limitations. If global sharing is done by software, data consistency and integrity must be guaranteed. This can affect performance.

Shared disk architecture is more I/O shipping oriented. (See 1.2.3.2, “I/O Shipping” on page 11.) I/O shipping provides more movement of data because the data is transferred before any operations are performed.

A third problem is data integrity. This problem arises when two or more processors try to update the same data. A global locking mechanism is needed. The lock can be at table level or at row level. The locking mechanism can be done by either hardware or by software. If it is done by hardware, then scalability and database size are limited. If locking mechanisms are implemented by software, performance may decrease since the work done by each CPU increases with the number of machines.

1.2.1.3 Shared Memory

With shared memory or shared all architecture, multiple processors access the same memory and disks. An optional local processor may be used as a buffer cache. Normally, with shared all implementations, there is only global memory and one set of disks, as shown in Figure 5.

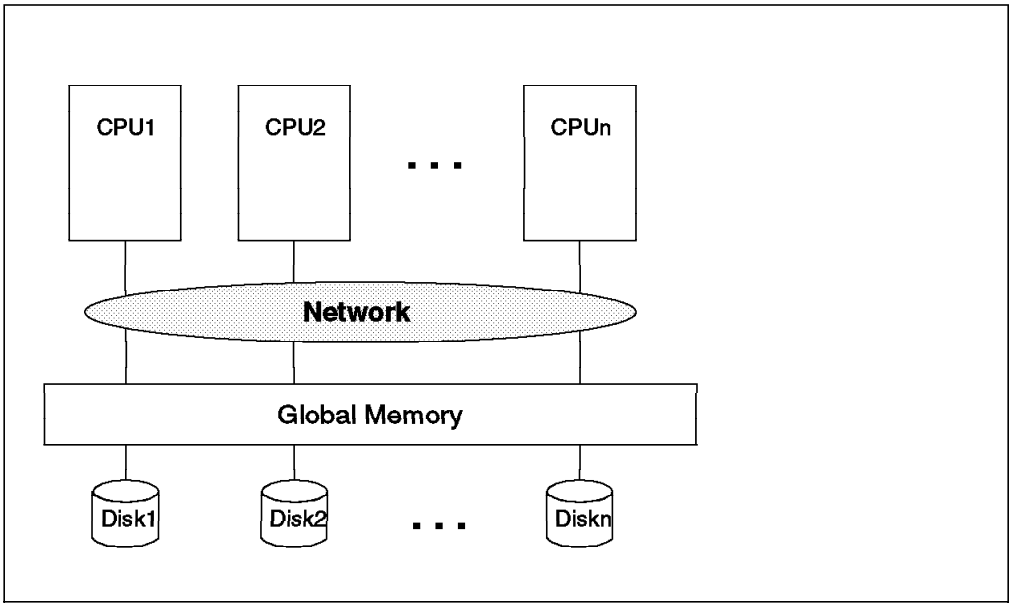


Figure 5. Shared Memory Architecture

Examples of machines that implement this architecture are:

- ES/9000 multiprocessor
- PowerPC SMP machines
- Other SMP machines

The main advantage of this architecture is performance. Since data is accessed locally by all the processors and memory is shared, the best performance results can be achieved in comparison to the other types of architectures.

The disadvantages of this architecture are the following:

- Limited scalability
- Limited memory

The advantage in performance is negated by the hardware limitations in the number of processors. Currently, the maximum number of processors available on the PowerPC Symmetric MultiProcessor (SMP) machines is eight.

Also, memory is limited in size and by the addressability of the software. This has a direct impact on database sizes. Performance is improved if more of the active database can reside in memory.

1.2.2 Parallel Processing

Transaction parallelism in database management is the way in which parallel processing is accomplished.

Transaction parallelism may be divided into two types:

- Inter-transaction parallelism
- Intra-query parallelism

1.2.2.1 Inter-Transaction Parallelism

Inter-transaction parallelism is achieved when different multiple transactions are executed simultaneously against one database. This is accomplished by having each available processor perform a different transaction. Consideration must be taken for the type of architecture that is implemented: shared nothing, shared disk, or shared memory.

Figure 6 illustrates inter-transaction parallelism.

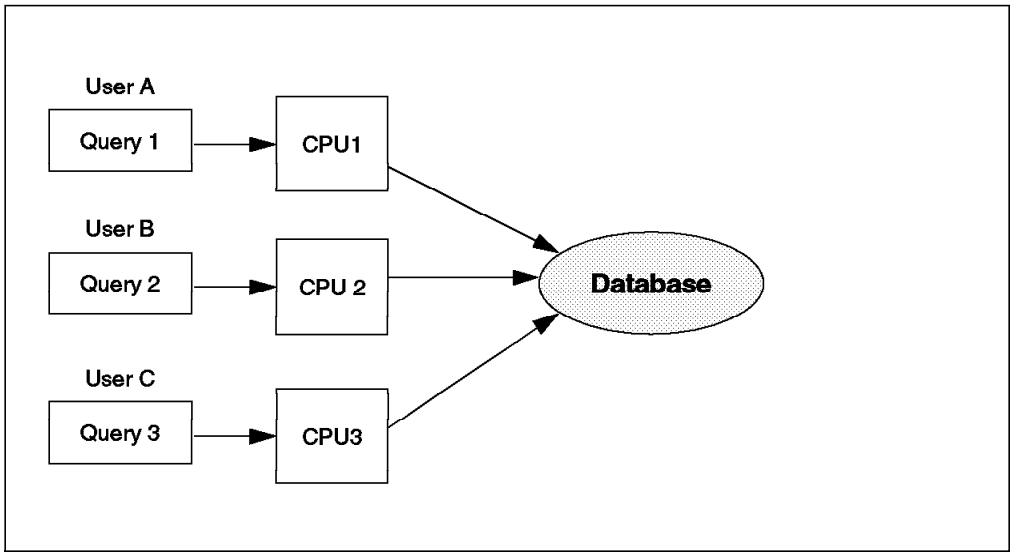


Figure 6. Inter-Transaction Parallelism

This type of parallelism is beneficial when there are many different concurrent transactions, none of which are heavily computational. Good results may also be gained if the size of the database is small but frequently accessed. Global

elapsed process time is reduced. The hardware architecture that fits best in inter-transaction parallelism is shared memory.

1.2.2.2 Intra-Query Parallelism

With intra-query parallelism, a single query is split across many processors. The benefit of intra-query parallelism is a speed-up in processing time. The elapsed time for performing a query may be reduced. This type of parallelism enables more complicated and/or more computational-intensive operations to be performed in a reasonable time span. This architecture is well suited to very large databases.

Intra-query parallelism can be achieved in two forms:

- Partition parallelism
- Pipelined parallelism

Figure 7 is a representation of partition parallelism.

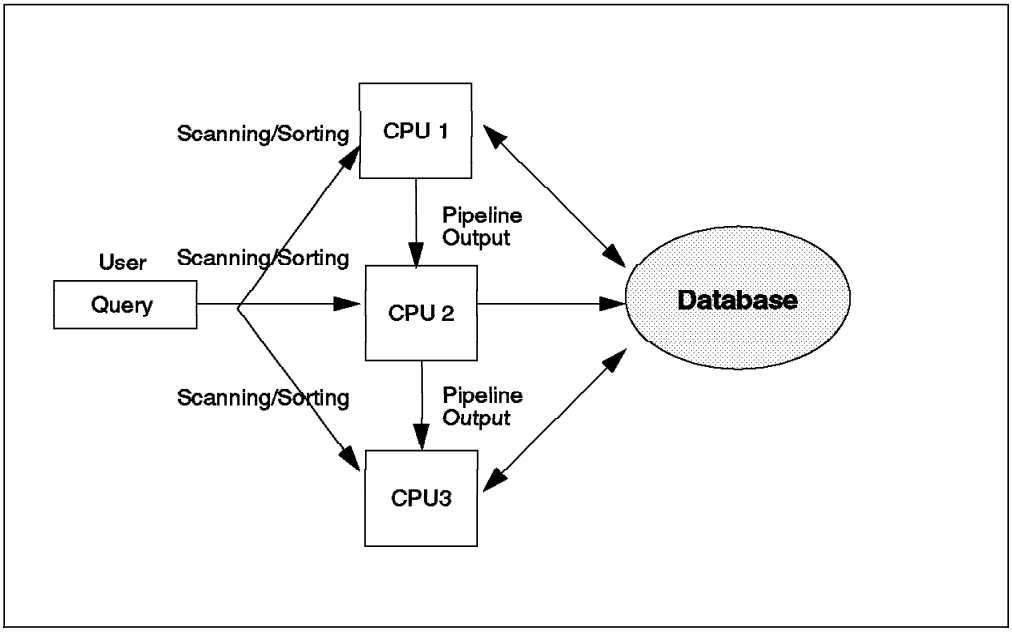


Figure 7. Intra-Query Partition Parallelism

Partition parallelism is also referred to as query decomposition. A single query is subdivided into several subqueries each of which processes a subset of the data.

Pipelined parallelism involves dividing the query into a series of operators. Figure 8 on page 10 is a representation of pipelined parallelism

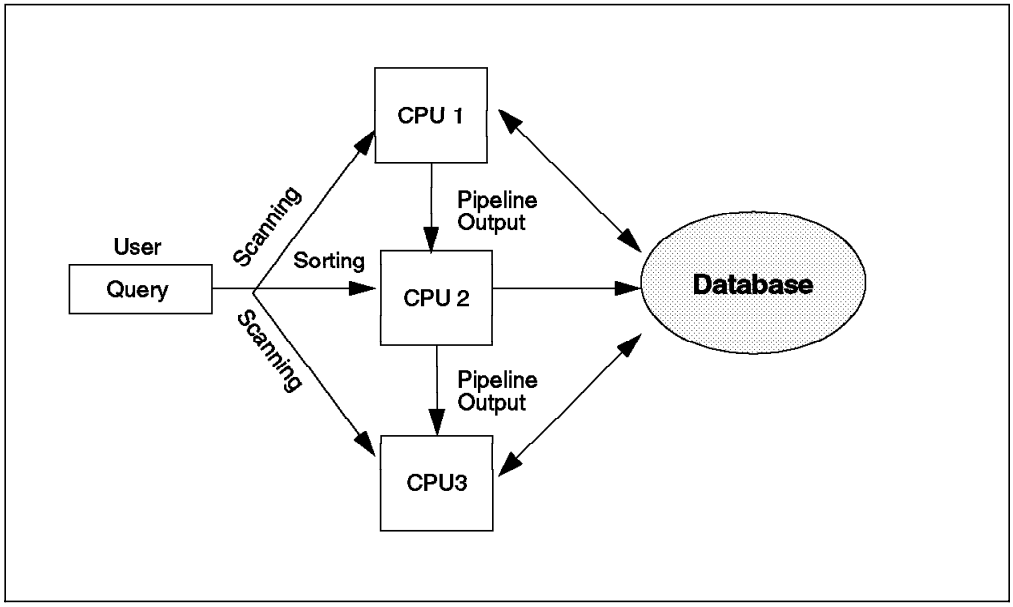


Figure 8. Intra-Query Pipelined Parallelism

These operators may be scanning and then sorting data. The output from one is used as input to the other.

1.2.3 Parallel Process Flow

Depending on the hardware architecture type, a parallel database can perform an operation in two ways:

- Function shipping
- I/O shipping

Function shipping splits the operations among the machines. I/O shipping ships the data to the machines for processing.

1.2.3.1 Function Shipping

Function shipping means that relational operators are executed on the processor containing the data whenever possible. So, the operation (or the SQL) is moved to where the data resides. Function shipping is well suited to the shared nothing architecture. Figure 9 on page 11 illustrates function shipping.

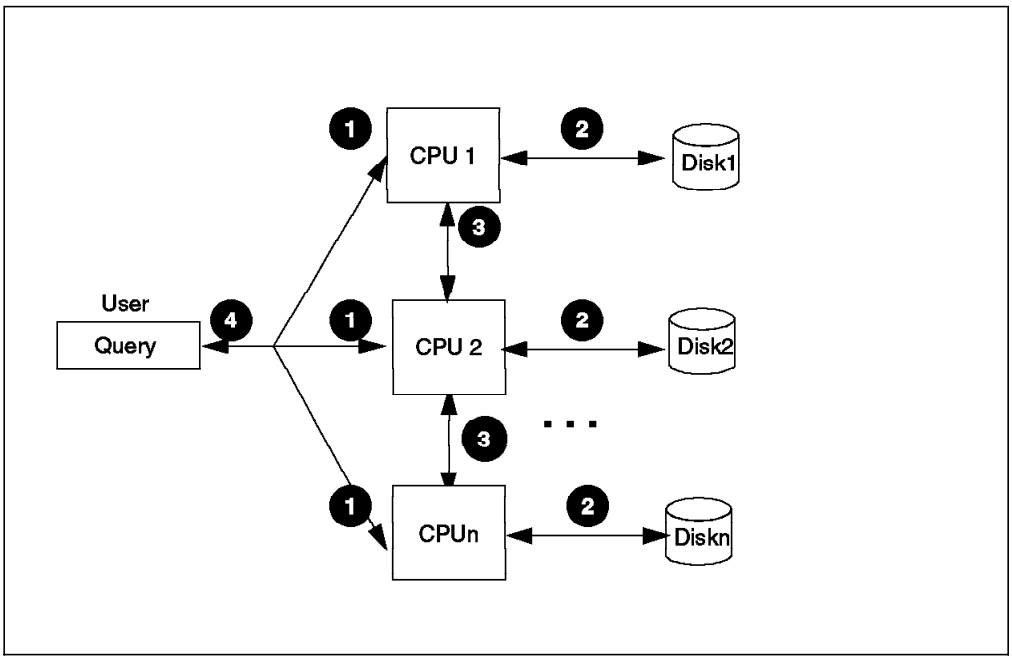


Figure 9. Function Shipping

The numbers in the figure are explained as follows:

1. A relational operator is invoked. In Figure 9, an SQL `select` is issued. Every processor receives the operation from one processor which works as a dispatcher. In DB2 Parallel Edition, this is called the coordinator node.
2. Every processor executes the operation on its own set of data.
3. An exchange of information among the nodes may occur.
4. The result from the operation is sent back to the coordinator node. The coordinator node assembles the data and returns it to the requestor.

This flow-control architecture has two advantages:

- It minimizes data communication.
- No central lock manager is needed.

Data transmission is minimized. Data is transmitted only if the operation requires it. There is no need for a central lock manager because every processor accesses only its own data. Only a global deadlock detector, checking at regular intervals, is needed.

This environment is well suited to a shared nothing architecture, possibly implemented on RS/6000 SP machines, either with or without HACMP Concurrent Resource Manager.

1.2.3.2 I/O Shipping

I/O shipping is implemented by data shipping to one or more processors and then executing the database operation. I/O shipping is particularly suited to a shared disk architecture.

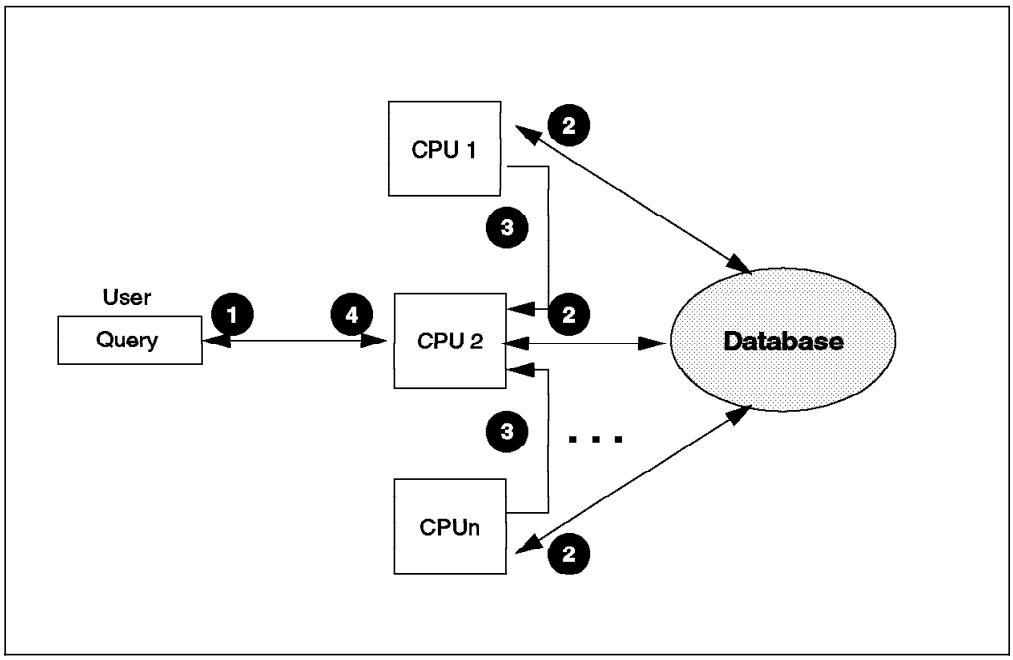


Figure 10. I/O Shipping Parallelism

Figure 10 illustrates the steps in I/O shipping:

1. A relational operator is invoked; for example, an SQL `select` is issued.
2. A processor is selected to run the operation. Figure 10 only shows one processor as being selected.
3. Every processor owning data involved in the operation sends it to the executing processor.
4. The executing processor performs the operation and returns the result to the requestor.

This flow-control architecture fits well in a heterogeneous environment. However, there are two disadvantages:

- Data movement is increased.
- Central lock manager is required.

The advantage of I/O shipping is that a system may be configured with specialized machines:

- I/O machines - These are machines that maintain a large number of disks.
- Compute machines - These are machines that have very limited I/O capacity.

While this type of configuration may fit specific needs, data communication is increased, and a global locking mechanism is necessary. In many cases, data is sent across the network that is not required for the operation. Moreover, since a piece of data can be requested at the same time by different processors, a central lock manager is needed. This limits the scalability of the system. If the system grows by adding a new processor, the work done by the central lock manager increases dramatically. This solution can be manageable if the database is not very big, and only a few I/O server nodes are necessary to keep all the data. However, with the increase of the database dimensions, the intrinsic limits of this architecture may soon be reached. Both an RS/6000 SP

using VSD and a cluster of RISC/6000 machines using the HACMP Concurrent Logical Volume Manager can fit into this environment by using I/O shipping.

DB2 Parallel Edition is an extension of Version 1 of the DB2/6000 database manager product. DB2 Parallel Edition implements a shared nothing architecture. The process flow architecture used is function shipping. Neither a central lock manager nor cross-system buffer invalidation mechanisms are necessary since data is only handled locally. These facts imply the following:

- The best performance is achieved in a distributed environment.
- Total scalability is permitted. There are no intrinsic limitations in hardware or software components.

Shared memory architecture may have faster data access than shared nothing architecture, but scalability is limited. Shared disk architecture has strong limitations both in performance and in scalability.

Chapter 2. Hardware Configurations

This chapter outlines the hardware configurations which DB2 PE supports, and suggests configurations which may be advantageous in various scenarios. The chapter is structured as follows:

- Network support
- Disk support
- RISC System Scalable Power Parallel System (RS/6000 SP)
- High-Availabilty support

2.1 Networks

RISC System/6000 machines can be interconnected via many types of networks. DB2 PE will run over any network which supports TCP/IP. This includes the following:

- Ethernet. This is a 10 Mb/sec network.
- Token-Ring. A token-ring can be configured to run at either 4 Mb/sec or 16 Mb/sec.
- Fiber Distributed Data Interface (FDDI). FDDI is an industry standard fiber-optic technology that provides high-speed (100 Mb/sec) communication among cluster members. FDDI provides built-in fault tolerance.
- Fibre Channel Standard (FCS). FCS is an industry standard fiber-optic switching network. It provides very high speed (266 MB/sec) connections between processors.
- IBM Serial Optical Channel Converter (SOCC). SOCC is a serial optical link that provides high-speed communication between two nodes. SOCC is still supported though no longer available for order.
- Asynchronous Transfer Mode (ATM). This is a type of packet switching that transfers fixed-length units of data.
- High-Performance Parallel Interface (HiPPI). HiPPI is an 800 Mb/sec interface to supercomputer networks, formerly known as high speed channel.
- High Performance Switch (HPS). HPS is a fast switching network that is available only in the RS/6000 SP machine. The High Performance Switch is available for MES upgrade orders only.
- Scalable POWERparallel Switch (SP Switch). The SP Switch is the next generation switch for the RS/6000 SP systems. Building on the same architecture as the High Performance Switch, it introduces improved reliability, availability, serviceability, and performance.

The current speeds of the various networks are given below:

Network	Speed
Ethernet	10 Mb/sec
Token-Ring	4 or 16 Mb/sec
FDDI	100 Mb/sec

Network	Speed
FCS	266 Mb/sec
SOCC	100 Mb/sec
ATM	155 Mb/sec
HiPPI	800 Mb/sec
HPS	35 MB/sec
SP Switch	70 MB/sec

Figure 11 shows the speeds of the different types of networks and how this speed is increasing over time.

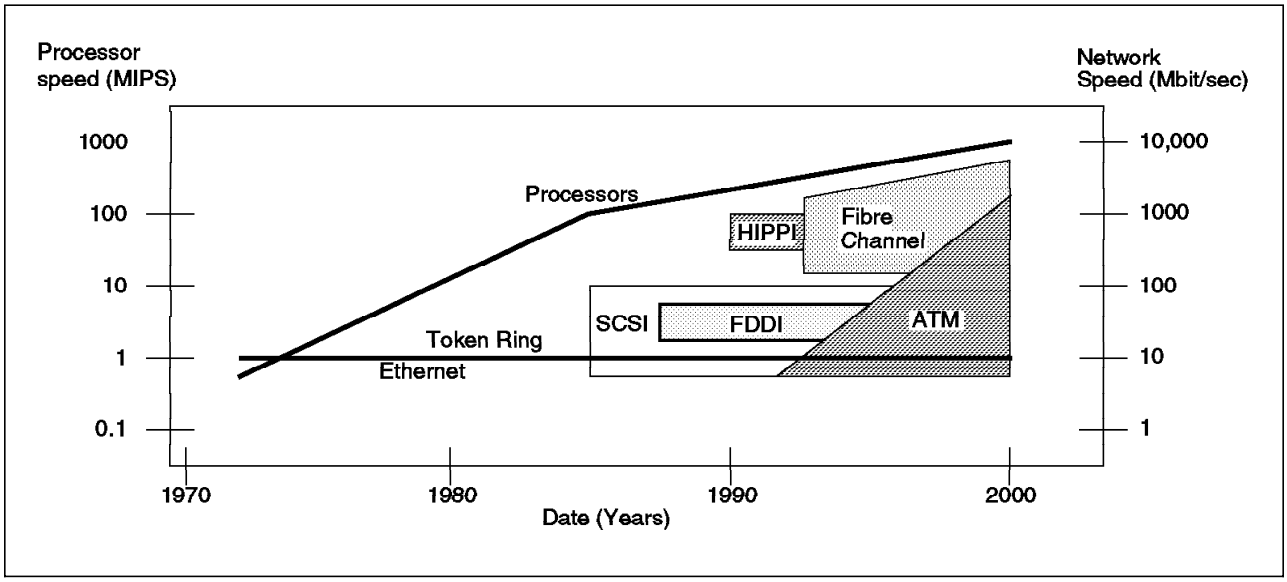


Figure 11. Network Speeds

Because DB2 PE is very dependent on the links between the machines, it is recommended that a private network be provided linking only the machines containing the DB2 PE database. For optimum performance, on any but very small clusters, this network is recommended to be FCS or FDDI. An Ethernet or token-ring network is likely to become a bottleneck to a DB2 PE cluster.

A network is also required between the clients and at least one machine within the DB2 PE cluster. The speed of this network may not be as critical as the private network, unless fetches containing a large number of lines are frequently required.

2.2 Disks

Data from DB2 PE is stored within a file system. This file system can reside on any type of RISC System/6000 disk. The choice of disk can be first split into two parts, internal disks and external disks.

2.2.1 Internal Storage

Internal disks are most often SCSI attached. There is always a limitation on the number of disks which can be attached internally. Table 1 shows the maximum capacities for each of the currently available systems.

Table 1. Maximum Internal Storage Capacities

RS/6000 System	Maximum Number of Drives	Maximum Drive Capacity (GB)	Maximum Total Capacity (GB)
Models 250/25T	1	2	2
Models 390/39H	1 + 2	4.5 + 9.1	22.7
Model G40	3	4.5	13.5
Models 3AT/3BT/3CT	1 + 3	4.5 + 9.1	31.8
Models 41T/41W	1 + 1	1 + 2	3
Models 42T/42W	2	2.2	4.4
Models C10/C20	3	2.2	6.6
Model 590/595	6	9.1	54.6
Model 591	1 + 5	4.5 + 9.1	50
Model 59H	2 + 4	4.5 + 9.1	45.4
Model J40	9	4.5	40.5
Model R20	1 + 1	2 + 2.2	4.2
Model R24	4	9.1	36.4
Model R40	1	2.2	2.2
Model H10	12 + 1	2.2 + 2.2	28.6

The current maximum size of an internal SCSI disk is 9.1 GB; so, for example, the maximum amount of internal disk which can be attached to a 590 machine is 54.6 GB. Space will probably be required on these disks to hold the AIX system and to contain paging space. This will reduce the amount of space available to store any data. In addition, you may choose to mirror file systems to improve resiliency, but this will further reduce the disk space available for data.

If more disk space is required on a machine, then external disks must be attached. External disks may also be chosen for reasons of speed. Serial disks are much faster than SCSI disks, but performance improvements may also be gained by using more than one SCSI adapter.

2.2.2 External Disks

External disks may be chosen either to improve disk access speed, to increase the amount of disk space attached to a machine, or for possibly both of these reasons.

Speed advantages may be gained in three ways.

1. Serial disks may be used. Data transmission to and from serial disks is much quicker than over SCSI.
2. A larger number of smaller disks may be used. This reduces the disk access time for an item of data. More than one disk may also return different parts of the same unit of data at the same time, which introduces parallelism.
3. More than one adapter may be used. Two SCSI adapters with disks attached provide two paths to data. This allows up to twice as much data per second to be accessed.

Three types of external disks can be attached to a RISC System/6000:

1. SCSI disks
2. Serial disks
3. Disk arrays

SCSI disks include a number of types of disk systems, including The 7204 External Disk Drive. These systems are relatively slow, but newer technologies, such as the SCSI-2 Fast/Wide, improves this.

Serial disks, such as the 7133 SSA Disk Subsystem, are much faster than SCSI disks, but they are more expensive. If speed is a priority, then these should be considered. Some serial disk adapters support RAID 5 to deliver data availability even in the unlikely event of a drive failure.

Disk arrays, such as the 7135 RAIDiant Array, consist of a number of small disks and a controller or controllers. Disk access is quite fast, but they are still limited by their SCSI attachment. Some models have two controllers and may be attached to more than one adapter. These can be active at the same time, potentially doubling the data transfer rate. Disk arrays can also provide data redundancy, which gives greater resiliency if a failure occurs.

2.3 RISC System/6000 Scalable Power Parallel System

The RISC System/6000 Scalable Power Parallel System (RS/6000 SP) is a family of scalable, parallel machines based on POWER, POWER2, and PowerPC 604 RISC architecture processors.

The components of the RS/6000 SP machine are:

- Processor nodes
- Internal networks
- Switch Network (optional)
- Control Workstation

2.3.1 Processors

The number of processor nodes in a standard RS/6000 SP configuration ranges from 2 to 128, which may be contained in multiple frames. Special bids allow up to 512 processors. Processors are contained in drawers that are packaged in frames. A single frame can contain from two to 16 processors.

Processors, or nodes, are available in three different types: thin nodes, wide nodes, and high (SMP) nodes. A processor drawer contains either one wide or two thin processor nodes. A high node occupies two full processor drawers. The RS/6000 SP thin node is typically configured as a compute node. It is functionally equivalent to a RISC System/6000 desktop system. It has half the width of a wide node, and can be packaged 16 to a frame.

The RS/6000 SP wide node is more often used as a server because it provides high bandwidth data access. It is functionally equivalent to a RISC System/6000 deskside system. It can be packaged up to eight per frame.

The RS/6000 high node is a symmetric multiprocessor (SMP) that contains a 2-way, 4-way, 6-way, or 8-way PowerPC 604 processor configuration. An RS/6000

high node is appropriate for customers requiring database and commercial processing capability. It can be packaged up to four per frame, and as few as one and as many as 16 PowerPC 604 processor node drawers can be installed in an SP system.

The RS/6000 SP system has seven processor node types. These are:

- Feature 2002— This drawer contains two POWER2 Thin Nodes running at 66 MHz. Each processor can hold up to 512 MB of memory, up to 9 GB of disk space, and has four Micro Channel slots. An optional 1 MB of Level 2 (L2) cache is offered.
- Feature 2003— This drawer contains one POWER2 Wide Node running at 66 MHz. It can hold up to 2 GB of memory, up to 18 GB of disk space, and has six available Micro Channel slots.
- Feature 2004— This drawer contains two POWER2 Thin 2 Nodes running at 66 MHz. Each processor can hold up to 512 MB of memory, up to 9 GB of disk space, and has four Micro Channel slots. An optional 2 MB of Level 2 (L2) cache is offered.
- Feature 2005— This drawer contains one POWER2 Wide Node running at 77 MHz. It can hold up to 2 GB of memory, up to 18 GB of disk space, and has six available Micro Channel slots.
- Feature 2006— This double drawer contains one PowerPC 604 High Node with a 2-way, 4-way, 6-way, or 8-way SMP running at 112 MHz. It can hold up to 2 GB of memory, up to 6.6 GB of disk space, and has 14 available Micro Channel slots.
- Feature 2007— This drawer contains one POWER2 Super Chip (P2SC) Wide Node running at 135 MHz. Each processor can hold up to 2 GB of memory, up to 18 GB of disk space, and has seven available Micro Channel slots.
- Feature 2008— This drawer contains two POWER2 Super Chip (P2SC) Thin Nodes running at 120 MHz. Each processor can hold up to 1 GB of memory, up to 9 GB of disk space, and has four Micro Channel slots.

This information is summarized in Table 2.

Table 2 (Page 1 of 2). RS/6000 SP Processors Summary

Feature Code	Clock Speed (MHz)	Memory (MB) std/max	Level 1 (L1) Cache instr/data	Level 2 (L2) Cache std/max	Internal Disk (GB) std/max	Micro Channel Slots
2002	66	64 MB/512 MB	32 KB/64 KB	0 MB/1 MB	1 GB/9 GB	4
2003	66	64 MB/2 GB	32 KB/256 KB	0 MB/0 MB	1 GB/18 GB	6
2004	66	64 MB/512 MB	32 KB/128 KB	0 MB/2 MB	1 GB/9 GB	4
2005	77	64 MB/2 GB	32 KB/256 KB	0 MB/0 MB	1 GB/18 GB	6
2006	112	64 MB/2 GB	16 KB/16 KB	1 MB/1 MB	2.2 GB/6.6 GB	14

Table 2 (Page 2 of 2). RS/6000 SP Processors Summary

Feature Code	Clock Speed (MHz)	Memory (MB) std/max	Level 1 (L1) Cache instr/data	Level 2 (L2) Cache std/max	Internal Disk (GB) std/max	Micro Channel Slots
2007	135	64 MB/2 GB	32 KB/128 KB	0 MB/0 MB	2 GB/18 GB	7
2008	120	64 MB/1 GB	32 KB/128 KB	0 MB/0 MB	2 GB/9 GB	4

Note:

- One Micro Channel expansion slot is used by the standard SCSI-2 High-Performance Internal Controller in the 66 MHz and 77 MHz Wide Nodes, and the 112 MHz High Node. One Micro Channel expansion slot is used by the required Ethernet High-Performance LAN Adapter in the 66 MHz, 77 MHz, and 135 MHz Wide Nodes, and the 112 MHz High Node.

A frame can hold up to 16 thin nodes, 8 wide nodes, or 4 high nodes. See Figure 12 on page 21 for an example of a frame containing 16 thin nodes with an Ethernet environment. See Figure 13 on page 22 for an example of a frame containing 8 wide nodes connected with Ethernet and token-ring. See Figure 14 on page 23 for an example of a frame containing a mixture of thin, wide, and high nodes.

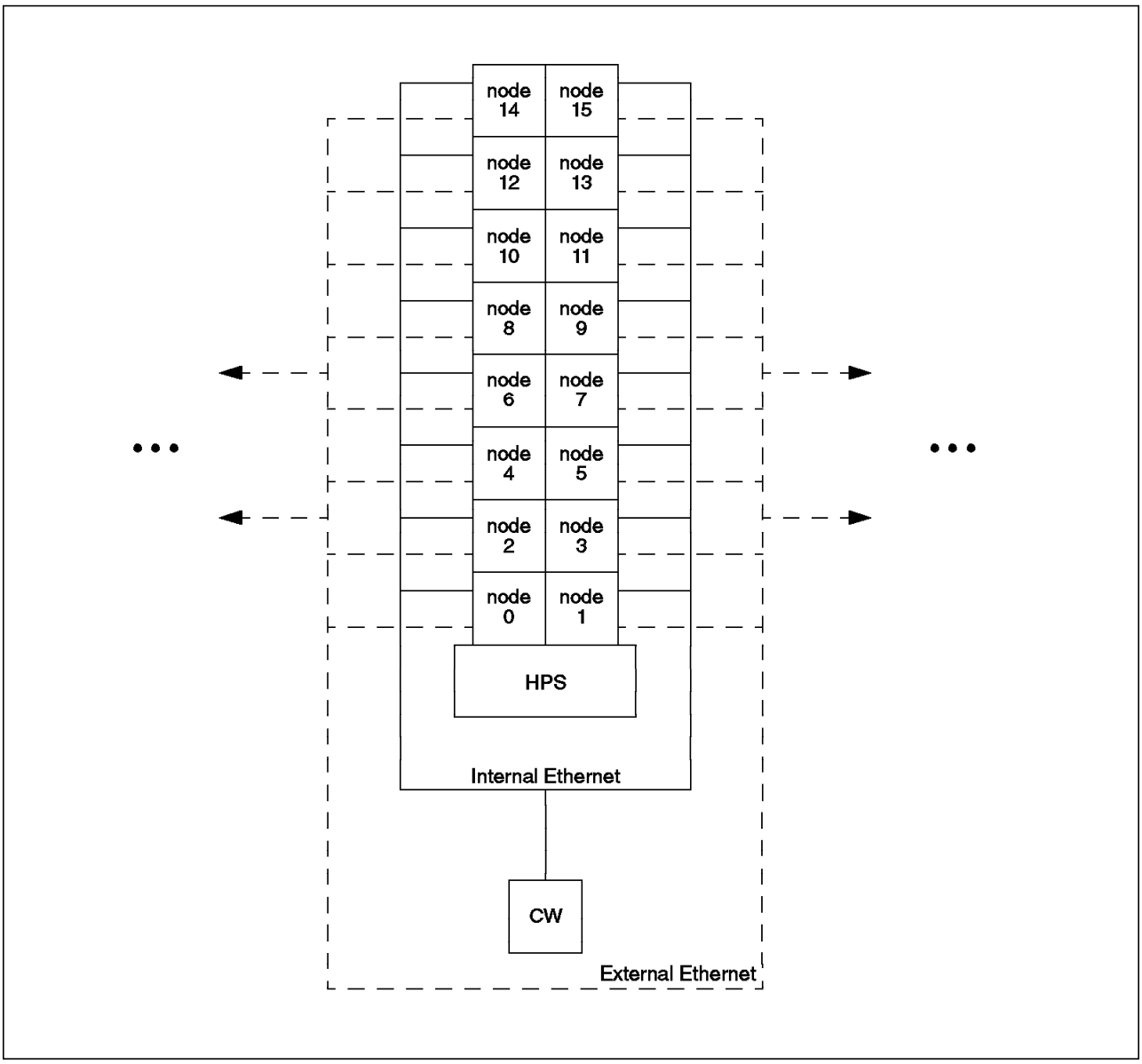


Figure 12. RS/6000 SP Hardware Configuration - Thin Nodes

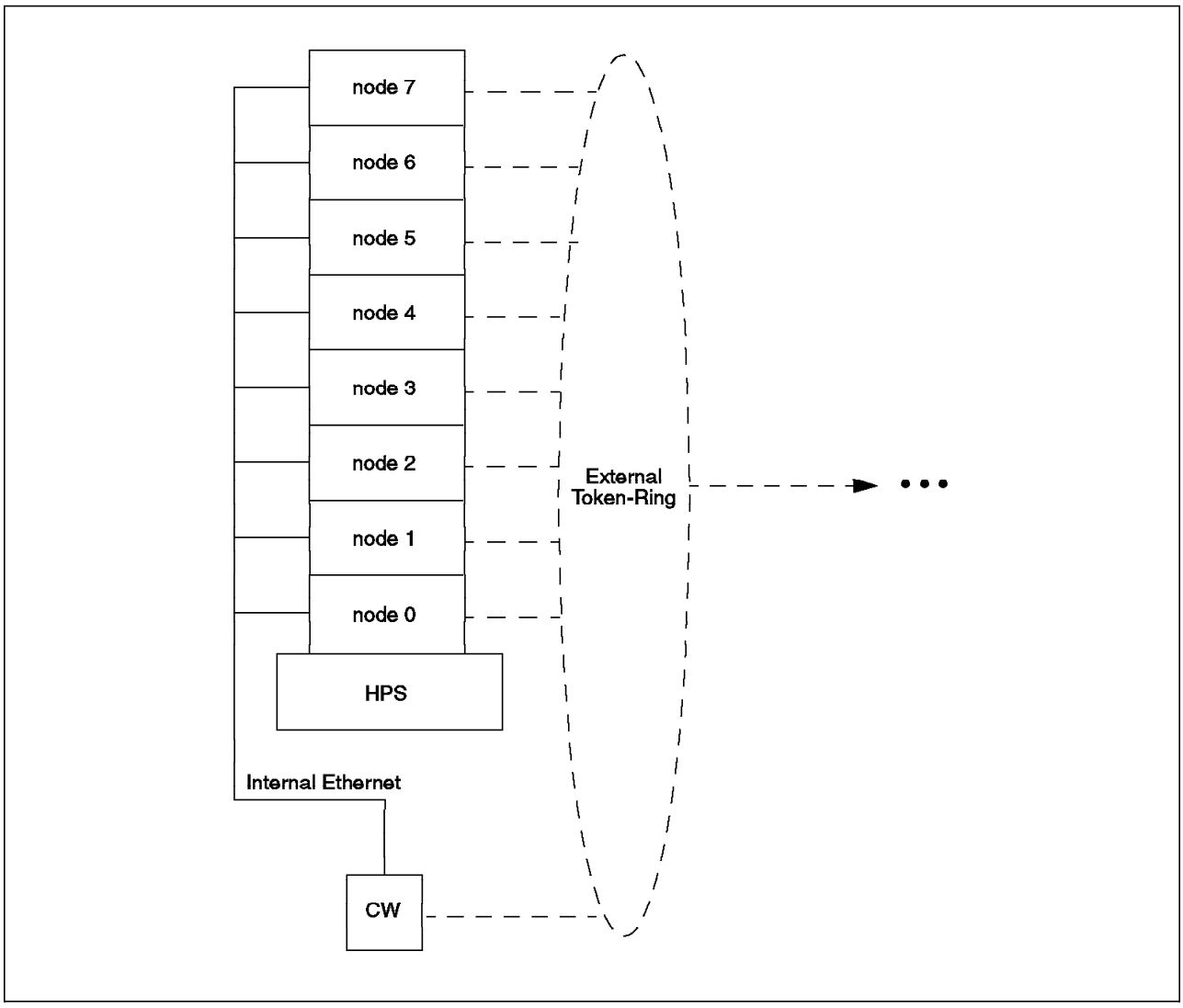


Figure 13. RS/6000 SP Hardware Configuration - Wide Nodes

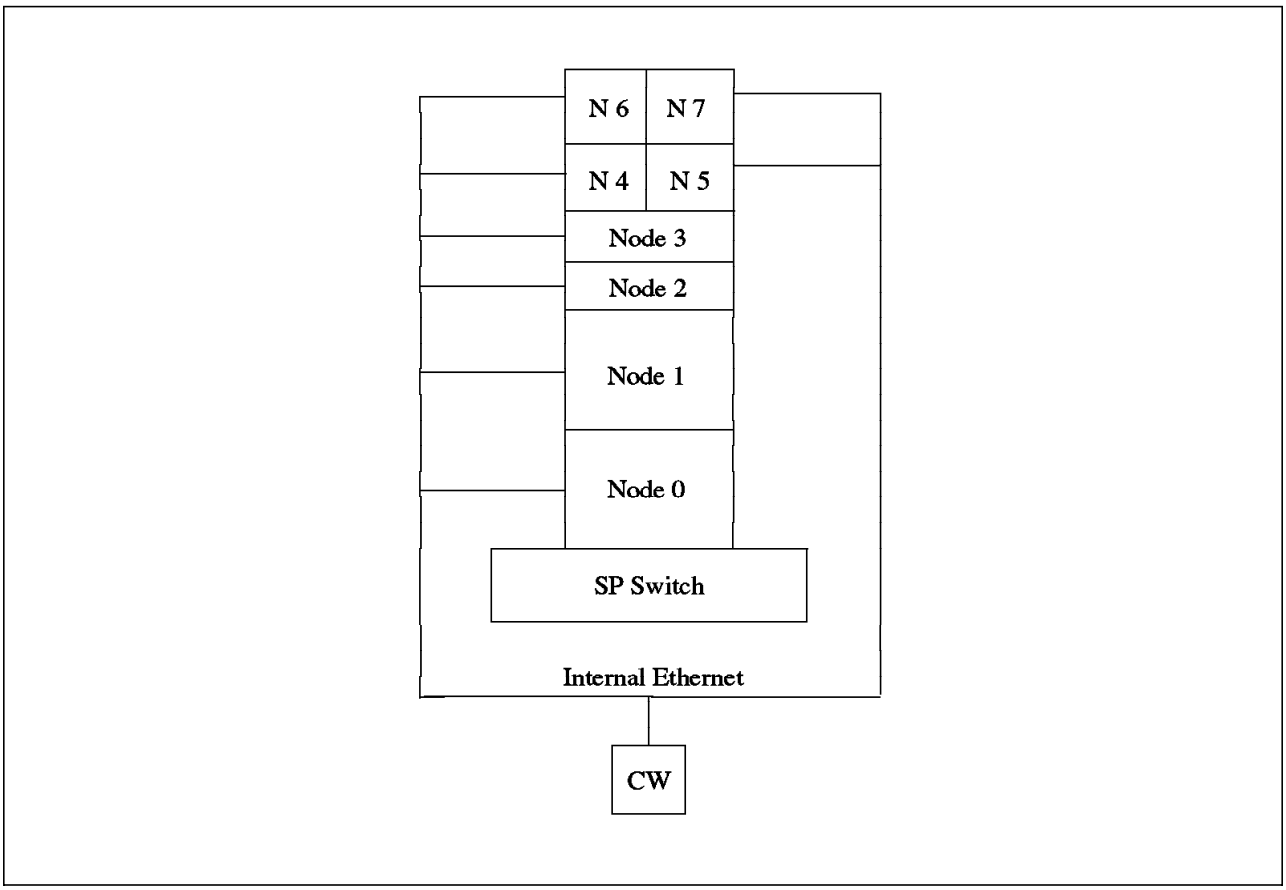


Figure 14. RS/6000 SP Hardware Configuration - Mixed Nodes

2.3.2 Networks

All RS/6000 SP processor nodes must have Ethernet and RS232 connections. The Ethernet network connects all nodes together and is used by the system-management software. This software controls installation, update and propagation of applications on the RS/6000 SP. The RS232 serial link connects each node to a control workstation. This link is used to monitor and manage the nodes in the machine.

Additional networks can also be added to an RS/6000 SP to improve performance, or to provide links to clients. The following networks are currently supported:

- Ethernet
- Token-Ring
- Fiber Distributed Data Interface (FDDI)
- Fiber Channel Standard (FCS)
- Serial Optical Channel Converter (SOCC)
- Asynchronous Transfer Mode (ATM)
- High-Performance Parallel Interface (HiPPI)
- Enterprise System Connection (ESCON)
- Block Multiplexor Adapter Card (BMCA)
- High Performance Switch (HPS)
- Scalable POWERparallel Switch (SP Switch)

2.3.3 Switch Network

The Switch Network is unique to an RS/6000 SP machine. It helps differentiate an RS/6000 SP from a networked group of RISC System/6000 workstations. The switch provides the message-passing network that connects all of the processors together in a way that allows them to send and receive messages simultaneously. The Switch Network is made up of two components, a central switch board and, in each node, a communications adapter. Figure 15 shows the RS/6000 SP Switch Network.

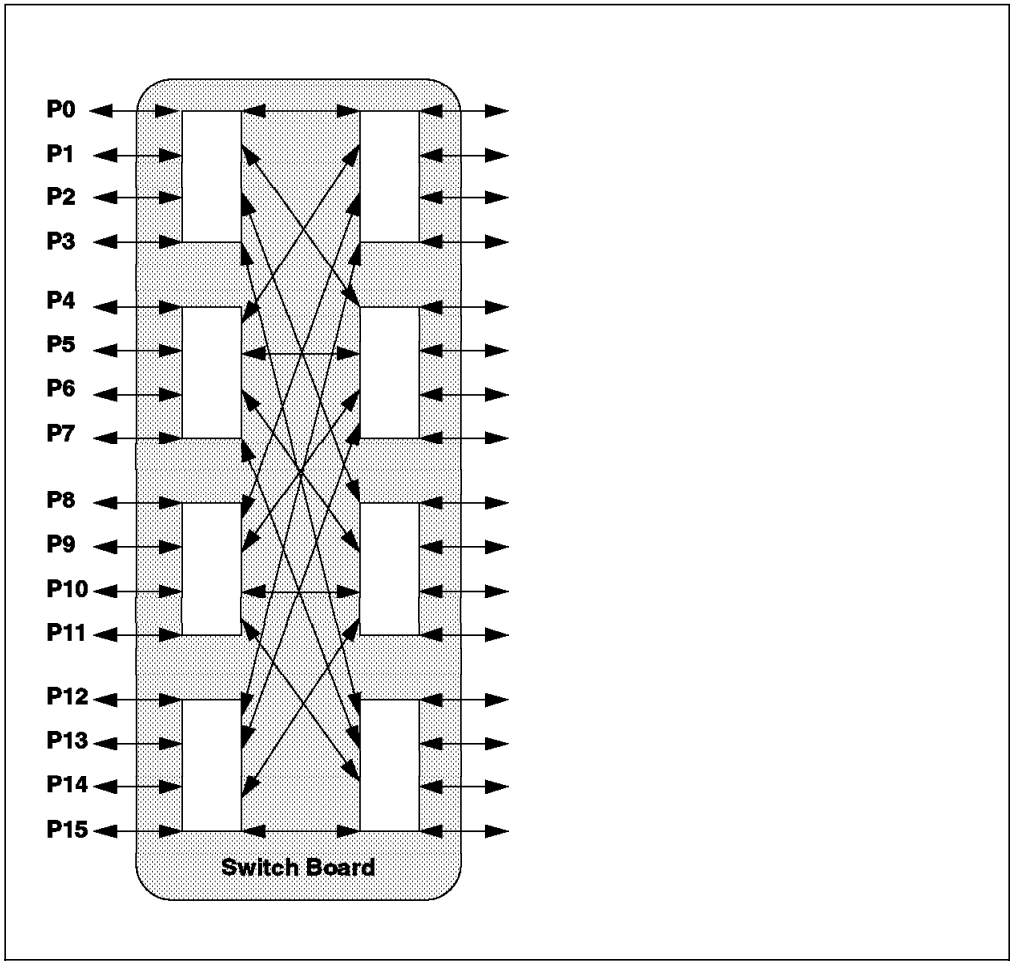


Figure 15. The RS/6000 SP Switch Network

The adapter has an on-board processor to handle network operations. This frees the node CPU from having to manage the transmission of data.

The switch board is a separate unit situated in the base of an RS/6000 rack. It is connected to each of the adapter cards and provides switches to link any pair of cards together. The formal definition of the switch is that it is an any-to-any, packet-switched, multi-stage, omega-type switch which has multiple paths between any pair of nodes. One switch board can connect up to 16 nodes. To connect more than 16 nodes, a switch cascade is used.

Figure 16 on page 25 shows four switch boards that are connected to 64 nodes across four RS/6000 SP frames.

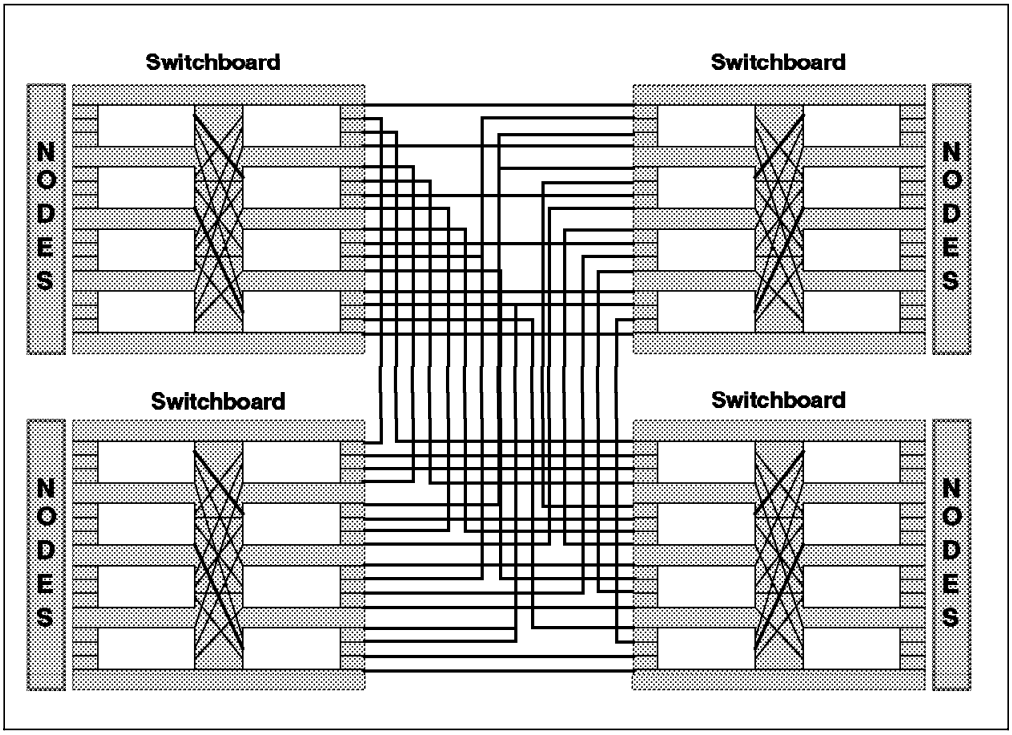


Figure 16. Switch Network - 64-Way

The switch is made up of one or more switch boards. A switch board provides two-stage 16X16 connectivity. Multiple boards are used to increase the connectivity by increasing the stages between any pair of communicating nodes. For systems with up to 80 processors, there is one switch per 16 processors. A frame can accommodate one switch board, besides the nodes in the frame. For systems with greater than 80 processors, extra switch boards are housed in a switch frame.

Each node connects to the switch board via a Micro Channel adapter and a cable to the switch board in the base of each frame. Wide-node-only frames have one switch per two frames since there is a maximum of eight wide nodes per frame. In a 604 High-Node-only configuration, up to four frames can share one switch.

Node outages are handled so that the switch continues to function, even when a node loses power or is disconnected. When a node is replaced or repaired, the switch can be reinitialized to include the new node without affecting jobs on other nodes. Transmission errors are monitored so that routes with excessive errors can be removed and replaced with working routes.

The advantages of the switch network are the following:

- Reliability

  For every node-to-node connection, a number of independent paths are available. This gives higher reliability and helps to avoid hot-spots.

- Communication Independence

  Communication independence means that any pair of nodes can communicate without interference from communications occurring between other nodes. Moreover, for every node-to-node connection, multiple

independent paths are available. This permits higher reliability and avoids hot spots.

- Flat Topology

  Flat topology means that the communication delay between any two nodes is the same. For example, the time required to communicate between nodes 1 and 2 or between nodes 1 and 123 takes the same amount of time.

  Flat topology offers two main advantages: node independence and scalability.

  – Node Independence

    The connection time between machines is almost identical. This means that a user could be moved from one set of nodes to another set without affecting his application performance. This allows workloads to be balanced easily. Also, the system manager does not have to select machines to be grouped according to the communication speed between them. This makes the management of the machine easier.

  – Scalability

    Scalability is also an advantage of flat topology. Adding nodes to the machine does affect the speed of interconnections and allows the machine to be upgraded easily.

For users who choose not to use a switch network, either Ethernet or FDDI may be selected for the communication between processors running parallel applications. However, this will affect the performance of the application.

2.3.4 Control Workstation

The Control Workstation is an external workstation that is connected to the RS/6000 SP machine by, at a minimum, Ethernet and serial RS232 lines. The Control Workstation is not required by the nodes except when software modifications are being done. It has two main functions, which are described below.

2.3.4.1 System Management Supervisor

The System Management Supervisor allows the Control Workstation to guarantee that the operating system installed on all the nodes is at the same level and that the same products are installed on all of them. In other words, it ensures that the environment is homogeneous.

The main idea behind the RS/6000 SP is to have the power of a parallel machine, with the usability of a single machine. To accomplish this goal, a number of generally available software products are utilized. These include Network File System (NFS), Andrew File System (AFS), Network Information Service (NIS), Network Time Protocol (NTP), timed and amd. In addition, extra software allows a system manager to install an application or do system maintenance once and automatically replicate the modifications to all of the nodes.

2.3.4.2 System Monitor Manager

The System Monitor Manager gives users a view of the state of the machine and its network status. It also allows an authorized user to control an RS/6000 SP machine. The state of the machine can be changed. For example, a node can be shut down very easily.

2.4 High-Availability Support

The High-Availability Cluster MultiProcessing/6000 (HACMP/6000) and High-Availability Cluster MultiProcessing for AIX (HACMP) products are used to provide high-availability services on the RISC System/6000. A set of system-wide, shared resources are utilized which cooperate to guarantee essential services.

HACMP/6000 Version 3 for AIX 3.2.5 and HACMP Version 4 for AIX Version 4 both support up to eight processors in a cluster. DB2 PE configurations can exceed these limits by spanning multiple high-availability clusters.

Processors are defined as cluster nodes to HACMP.

2.4.1 Concurrency

Data can be accessed concurrently or non-concurrently under HACMP . Under concurrent access, more than one machine accesses the same disk at the same time. Concurrent access is not used by DB2/6000 or by DB2 PE.

2.4.2 Points of Failure

For an HACMP cluster to be effective, single points of failure should be eliminated. A single point of failure exists when a critical cluster function is provided by a single component. If that component fails, the function can no longer be provided, and an essential service becomes unavailable.

Examples of cluster components which are potential single points of failure include:

- Processors
- Power sources
- Network adapters and networks
- Disk adapters and media
- Control workstations

2.4.2.1 Processors

Processors can be eliminated as a single point of failure by having standby processors ready to take over their workloads should they fail. These standby processors can be configured with HACMP to behave in three different ways:

1. Idle Standby—A standby processor can be provided that will take over the work of a failed processor. When the failed processor is fixed and reintegrated into the cluster, it will reclaim its resources. The standby processor must have access to all resources required for the provision of the essential services — disks, networks, and so on. This method results in a cluster which will not lose performance after a failure, provided the standby processor has the same capacity. The disadvantage is the cost since the standby processor is not used except after a processor failure.

2. Rotating Standby—A standby processor is provided to take over of a failed processor, as in the idle standby scenario. However, when the failed processor is reintegrated, it does not reclaim its resources, but becomes new standby machine. This configuration has the same cost characteristics as idle standby. Its advantage is that it avoids the impact of the originally failed processor reclaiming its resources when it rejoins the cluster.
3. Mutual Takeover—There are no standby processors; all processors are utilized in a normal state. After a processor failure, the failed processor's resources and essential services are taken over by one of the surviving processors in addition to its normal services. This method uses hardware resources more efficiently because redundant processors are not used. The disadvantage is that there may be performance degradation after a processor failure.

2.4.2.2 Power Sources

Either uninterruptable power supplies, dual power sources, or both should be used.

2.4.2.3 Networks

Secondary networks should be available to cope with the failure of the primary network.

Standby network adapters are used to cope with a processor failure in a mutual takeover or hot standby cluster. After a processor failure, the processor which takes over the failed nodes services will be required to have two network addresses: its own network address and the address of the failed processor. To do this, two adapters are required for each machine. The standby network adapter can also take over the machine's primary address if the primary adapter fails.

The following TCP/IP networks are supported in an HACMP environment:

- Ethernet
- Token-Ring
- Fiber Distributed Data Interchange (FDDI)
- Fiber Channel Adapter
- Asynchronous Transfer Mode (ATM)
- High-Performance Switch (HPS)
- Scalable POWERParallel Switch (SP Switch)

It is strongly recommended that all nodes sharing an external disk have point-to-point, non-TCP/IP connections. Should TCP/IP fail, HACMP will still be able to communicate via the non-TCP/IP connection to prevent spurious disk takeovers. The following options are available:

- Raw RS-232 link
- SCSI-2 Differential bus

2.4.2.4 Disks

Disks can be attached to more than one processor. This allows a disk to be accessed by a takeover processor. Data held on disks other than RAID disks should be mirrored to prevent loss of data after a disk failure. To ensure no single point of failure, these mirrors should be accessed via different adapters.

Disks which are supported include external SCSI disks and enclosures, Serial disk subsystems, and various RAID subsystems.

- SCSI — SCSI-2 Differential disk subsystems should be used. A chain of SCSI disks can connect to up to four nodes. Each disk is only "owned" by one node at a time. This ownership can be transferred after a failure. This solution has the benefit of lower cost, but access is relatively slow. Because access time is important, these disks will probably not be appropriate.
- 9333 Serial Disk Subsystems — These provide increased system throughput, disk capacity and performance. Models 9333-010 and 9333-500 can connect to two processors. Models 9333-011 and 9333-501 can connect to eight processors.
- 7133 SSA Disk Subsystem — IBM 7133 Serial Storage Architecture Disk Subsystem Models 020 and 600 are second-generation, leading-edge serial storage subsystems that implement industry-standard Serial Storage Architecture (SSA) to deliver outstanding performance, capacity, and availability in a low cost, scalable package. Combined with IBM′s family of SSA adapters, the 7133 Models 020 and 600 can be configured for high performance, multihost attachments, or economical RAID 5 data protected storage.
- 7135 RAIDiant Array — This is a disk array controller with an SCSI-2 differential fast/wide host interface and multiple SCSI busses which have attached disk drives. The array supports RAID levels 0, 1, 3, and 5. RAID levels 1, 3 and 5 offer data redundancy, so mirroring is not necessary.

  Under level 0, each group of disks is used without any form of data redundancy. This provides the most storage, but does not offer high availability.

  With level 1, all disks are mirrored on a one-for-one basis within a group. This provides the fastest read access, and the best performance in case of a failure, but it is costly.

  For level 3, one disk within the group is used for parity, and the other disks hold the data. The parity disk contains sufficient information about all the other disks in the group to be able to rebuild any data after the loss of any single disk. After a disk failure, no data is lost, but the time taken to access the data increases considerably.

  Level 5 is similar to level 3, except the parity data is striped across all disks within the group. This provides faster access than level 3. Level 5 provides fast write access, but read access is slower than level 1. It has very good price/performance characteristics.

  Level 1 and level 5 are the two RAID levels which should be considered for DB2 PE. If cost is an issue, then level 5 is recommended. If optimum performance is needed, and this must be maintained after a failure, then level 1 should be considered.

  7137 High Availability External Disk Array — This provides functionally similar to the 7135 RAIDiant Array. It only contains one controller, but allows hot

standby disks to be configured. After a disk failure, the data from the failed disk will be recreated onto the standby disk. If a second disk failure occurs, then the parity disk can be used to provide continuous service.

3514 Disk Array Subsystem. This is a cheaper option. It provides RAID levels 1 and 5, and only has one controller.

2.4.2.5 Control Workstation

An optional High Availability Control Workstation (HACWS) connectivity feature allows a backup control workstation to be connected to an SP system. The backup control workstation is utilized when the primary control workstation is unavailable due to scheduled software upgrades or unscheduled outages during normal operation.

2.4.3 Networked Systems

Networked systems are groups of RISC System/6000 machines loosely connected via at least one network.

There are numerous possible configurations which are possible with HACMP and DB2 PE. Any implementation will have specific requirements which may well dictate the properties of the implemented HACMP cluster. Two examples of networked clusters that will provide good solutions in many cases are given in this section. The first should provide excellent price performance, but would suffer degradation after a failure. The second is a more costly solution, but performance after a failure should have little degradation.

2.4.3.1 Example 1 — Mutual Takeover Scenario

The first example is of a mutual takeover scenario.

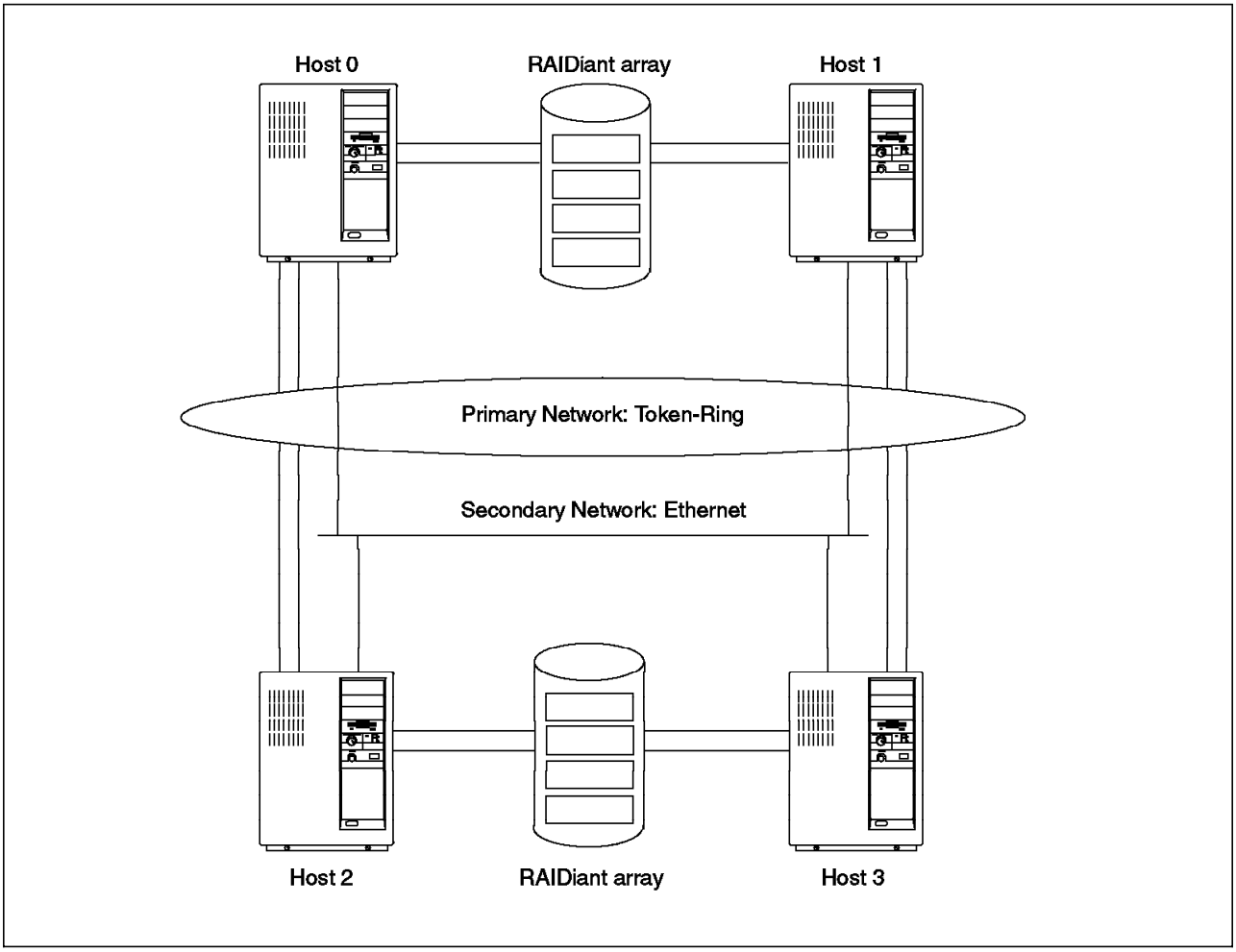


Figure 17. HACMP Mutual Takeover Cluster Example

Figure 17 shows an example which uses RAIDiant arrays as the storage method for the database. The systems are connected via a primary 16 Mb token-ring and a secondary Ethernet network. The primary network has standby adapters to allow for takeover.

The connections to the RAIDiant arrays should be duplicated in each machine to protect against a SCSI adapter failure or SCSI cable failure. RAID level 5 would be implemented in the disk groups.

The cluster consists of two mutual takeover pairs of machines. Each pair has access to a shared set of RAIDiant arrays. In a normal situation, some of the disks will be used by one of the nodes and some by the other. If a node fails, its partner will take over the rest of the RAIDiant array disks and the hardware IP address, and will start up the failed node′s instance of DB2 PE. For example, after a failure of Host 0, Host 1 will take over the resources from Host 0 and start the instance of DB2 PE from Host 0.

After a node failure, the database will be running on three nodes instead of four. This will probably result in a severe degradation in performance.

After a disk failure, the RAIDiant arrays will continue to provide uninterrupted access to data; however, access time will increase significantly since the data will have to be extracted using the parity information.

After a network failure, the machines will use the backup Ethernet network.

This configuration provides a system which is very cost-effective. During normal operation, it should provide an optimum database system for the number of nodes. However, after a failure, although the database will continue to be available, the performance is likely to degrade significantly.

2.4.3.2 Example 2 — Rotating Standby Scenario

The second example is of a rotating standby scenario.

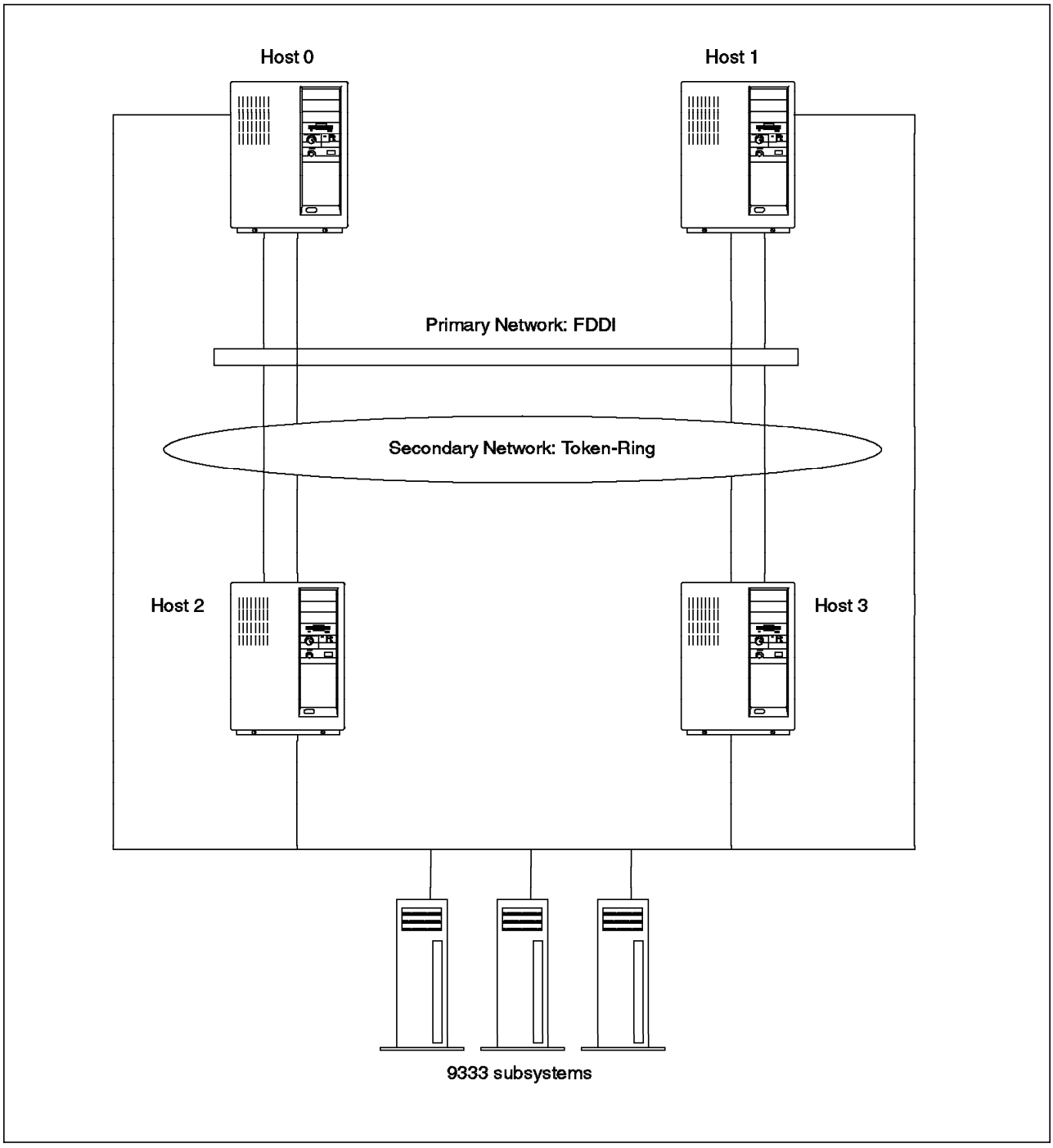


Figure 18. HACMP Standby Cluster Example

Figure 18 shows an example where a cluster uses a 7133 SSA disk subsystem as the storage method. The nodes are connected via a primary FDDI network and a secondary 16 Mb token-ring.

Three IP addresses and three database partitions are defined for the cluster. The first three nodes to be started will control these resources, and the last node will be a standby node. All the nodes are connected to all the 7133 subsystems to allow them to access the disks for the database partition they are running.

The data held on the 7133 subsystems should be mirrored to protect against failure. These mirrors should be across different 7133 subsystems and different adapters, and therefore on different SSA loops, to ensure no single point of failure.

Raw RS232 null modem connections could be provided between all nodes. This would be used to prevent a node from attempting to take over the disks after a TCP/IP failure.

After an FDDI adapter failure, the cluster would be configured to run a node down script to allow the instance to move to a standby node.

The secondary, token-ring network is used in the event of a failure of the primary network.

This example illustrates a system that should provide a high level of performance. It should also provide a system that has little or no degradation of performance after a failure.

2.4.4 RS/6000 SP Configuration

This section provides an example configuration with four nodes.

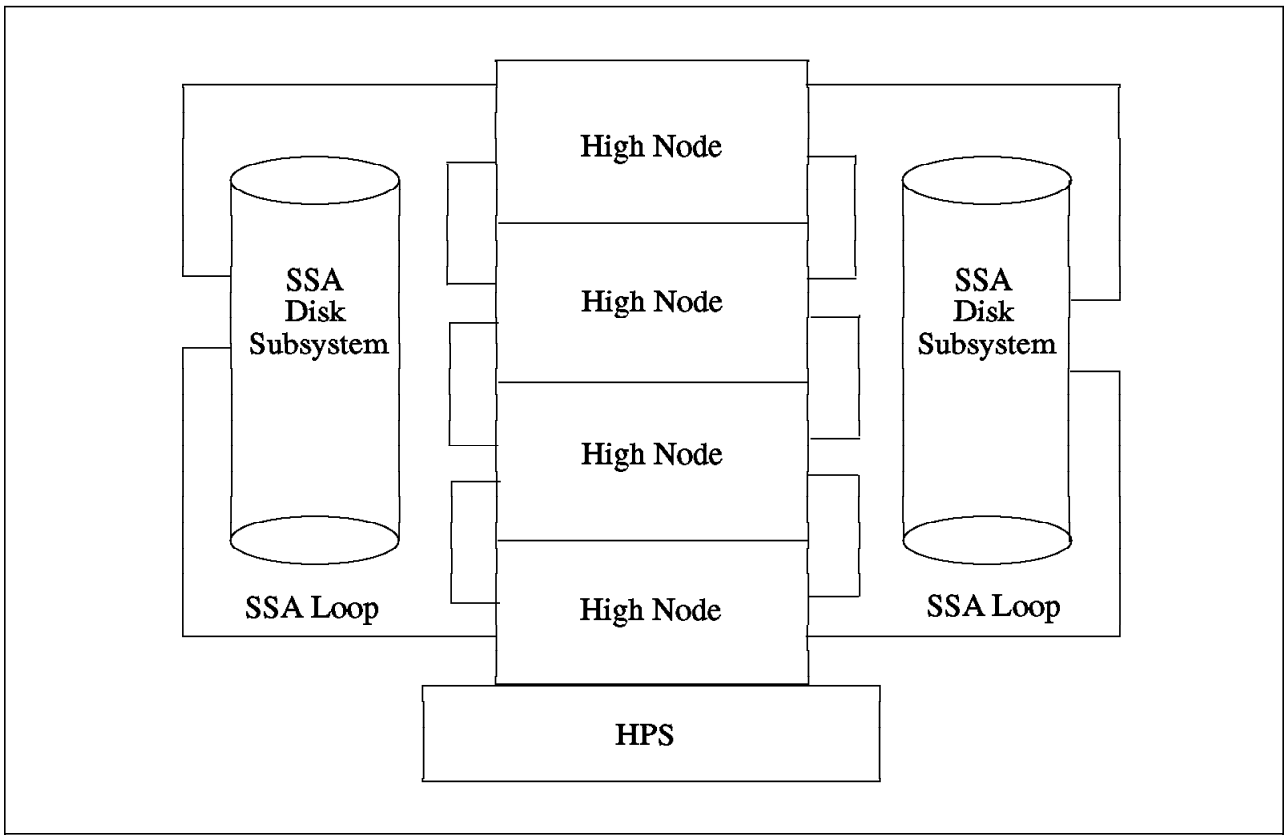


Figure 19. RS/6000 SP Configuration with HACMP

Each of the four high nodes contains:

- 1 SP Switch adapter
- 1 Ethernet adapter
- 2 Disk adapters
- 2 SSA adapters

All the nodes run DB2 PE. The nodes use the SP Switch as their primary network and Ethernet as the secondary network. The data is held on 7133 SSA disks. The data can be mirrored across adapters, or can be protected by SSA RAID adapters, depending upon requirements.

The nodes are set up as four clusters of two nodes. Each cluster is a mutual takeover cluster. After a failure, the backup node may vary on the volume groups of the failed node and start the failed DB2 PE partition.

Chapter 3. Concepts and Data Placement

This chapter discusses the placement of data within a DB2 Parallel Edition database with regard to the setup and configuration of the parallel database environment. Some new terminology will be introduced in this chapter as well. This chapter is outlined as follows:

- Parallel Database
- DB2 Parallel Edition Architecture
- DB2 Parallel Edition Process Model
- Parallel Database Nodes
- Database Instance
- Database Management and AIX
- Segmented Tables
- Multi-Page File Allocation
- Nodegroups and Data Partitioning

3.1 Parallel Database

A database is simply a collection of data. The data is managed by a database manager. The database manager controls CPU, memory, disk, and communications resources. The database manager also provides users with the ability to store and access the data. The collection of data and system resources that is managed by a single database manager, together with its database manager software, is referred to collectively as a node. The node resides on a processor, and one or more nodes can be assigned to each processor. For a serial database, a single node is assigned to a single processor. In DB2 Parallel Edition, a node is called a Parallel Database Node or PDB node.

A parallel database is one that contains several nodes assigned to one or more processors. In a shared-nothing parallel database system, like DB2 Parallel Edition, nodes are assigned to several processors. The processors share neither disk nor memory, but they communicate with each other via messages. The end users are not aware that the database may be split across processors.

The database manager at each node of the parallel database system is responsible for its part of the database′s total data. The data is not duplicated across the nodes, instead each node owns a part of the database. This structure is known as a database partition.

Data is usually distributed across nodes in such a way as to ensure that each node has approximately the same amount of data. By dividing the database across many nodes, the power of multiple processors can be exploited to satisfy complex requests for information. Suppose you have 100 million records on a serial database. To scan all the records would require a single node to look at all 100 million records. Now suppose the same records were spread across twenty nodes. Each node would only need to scan around 5 million records, and if all processors performed their scans at the same time, the result would be available sooner.

Applications connect to the database via an application coordinator node. With DB2 Parallel Edition, any node can act as an application coordinator node. You may wish to consider spreading out users across nodes to distribute the coordinator function.

3.2 DB2 Parallel Edition Architecture

DB2 Parallel Edition, an extension of the DB2/6000 Version 1 product, is a parallel database product that operates on AIX-based parallel processing systems, such as the IBM RS/6000 SP. DB2 Parallel Edition supports a shared-nothing architecture. In the shared-nothing architecture, each processor has its own memory and a pool of disks. For more information about the shared-nothing architecture, see 1.2.1.1, “Shared Nothing” on page 4. DB2 PE uses the function-shipping execution model for its parallel process flow. In function shipping, database operations can be executed on the processor where the data resides. For more information about function shipping, see 1.2.3.1, “Function Shipping” on page 10. The architecture and implementation of DB2 Parallel Edition provides scalability and capacity. It can support and maintain very large databases from hundreds of gigabytes to terabytes of data.

3.2.1 Major Components

The system architecture of a parallel database node consists of the basic components of DB2/6000 V1, plus extensions to support parallel execution. The following provides a high-level description of these extensions:

- Data definition language (DDL)

  The DDL is extended to support concepts such as nodegroups and table partitions. New DDL, `create nodegroup`, is introduced to support partial declustering, overlapped assignment and hash partitioning of data tables. The `create table` DDL is extended to support partitioning keys and nodegroups.

- Utilities

  The utilities and tools are extended to manage the parallel database system. There are a variety of new utilities, such as data loading, adding and deleting nodes, and data redistribution.

- SQL compiler

  The SQL compiler is extended to support a parallel plan. A parallel plan is one that determines the best parallel query execution strategies and produces the best parallel access plans for different types of SQL queries. During the compilation process of each SQL request, the SQL optimizer makes use of the data distribution and partitioning information of the base tables from the system catalog to assist in selection of parallel execution strategies. The choice of the optimal plan is based on least cost estimation.

- Control services

  This component is extended to handle and manage the interprocess control message flow, such as start and stop processes, error reporting, and interruptions.

- Table queue services

  This component handles the exchange of rows between DB2 Parallel Edition agents across or within a node.

- Communication services

  This component provides the internodal communication via a fast communication manager (FCM) process. It is also responsible for the distribution of the parallel requests.

- Data protection services (DPS)

  This component is extended to support global deadlock detection, two-phase commit, and recovery between nodes.

3.2.2 Parallel Edition Process Model

The following major processes are created on a parallel database node during the startup of the database manager with the default database manager configuration parameters:

- System controller (db2sysc)

  This process handles system initialization during db2start, agent initialization and system shutdown.

- Watch dog (db2wdog)

  This process monitors the other database processes in the system and cleans up resources during abnormal termination.

- Parallel system controller (db2pdbc)

  This process manages the parallel agent pool and handles parallel requests.

- Fast communication manager (db2fcmdm)

  This process handles messages between agents on the same or different nodes.

- Master database logger (db2loggr)

  This process manages the logging of the database to maintain data integrity. During the first connection to a database, this process spawns a child logger process (db2lwrt).

- Master database deadlock detector (db2dlock)

  This is the master deadlock detector. During the first connection to a database, this process spawns a global deadlock detector process for the coordinator node and a local deadlock detector for each node.

Figure 20 on page 38 illustrates the parent-child processes relationship which is created during the startup of the database manager using default database manager configuration parameters on a multi-node system.

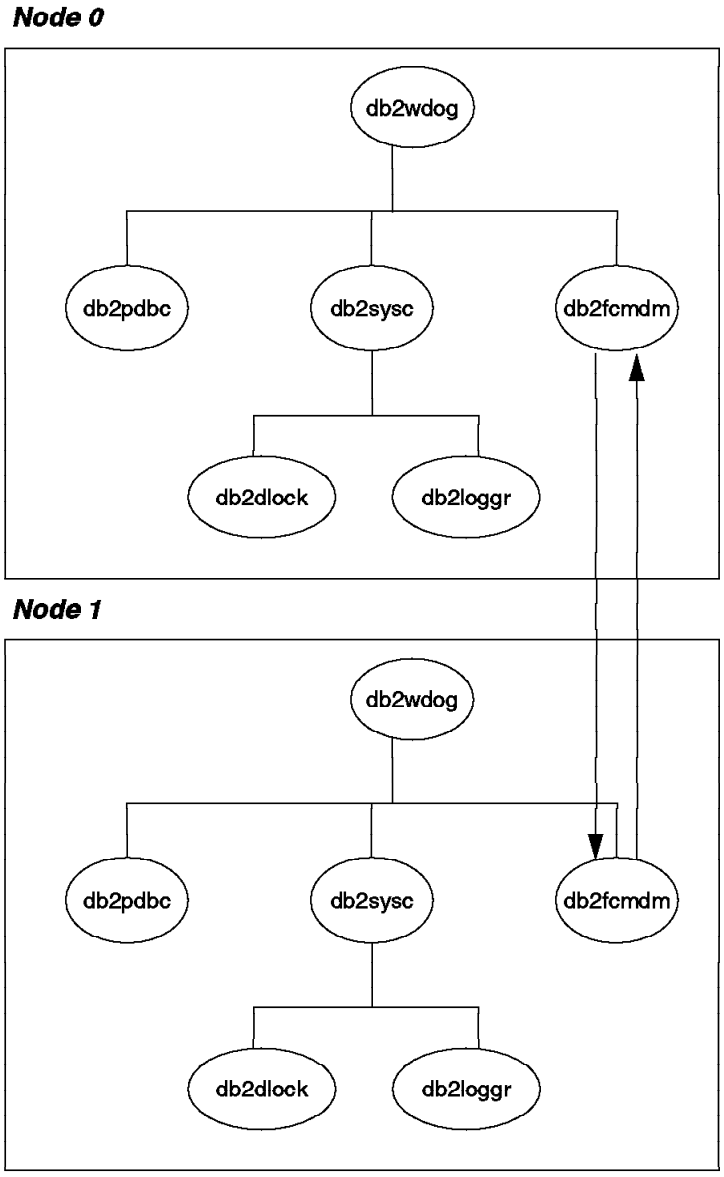


Figure 20. Parent-Child Processes Relationship During Startup

After the database manager is started, and the database is created on the system, the system is ready to accept database connection requests. During the first database connection, the following major processes are created on the catalog and coordinator nodes:

- Coordinating agent (db2agent)

  This process is only created at the coordinator node. It handles the SQL process from a user request. The maximum number of coordinating agents which can exist at one time on a node is controlled by the MAX_COORDAGENTS parameter in the database manager configuration.

- Global deadlock detector (db2glock)

This process is only created on the catalog node. It collects locking information from the local deadlock detector processes and handles global deadlock detection for the parallel database system.

- Local deadlock detector (db2llock)

  This process handles the deadlock detection for the local node.

- Child database logger (db2lwrt)

  This process handles the logging I/O.

Figure 21 on page 40 illustrates the parent-child processes relationship which is created during the first database connection. In this example, the `create database` command is executed on node 0, and the user request is connected to node 0. This makes node 0 both the catalog node and the coordinator node. The firewall shown in Figure 21 on page 40 is used to protect the database and database manager from errant database applications.

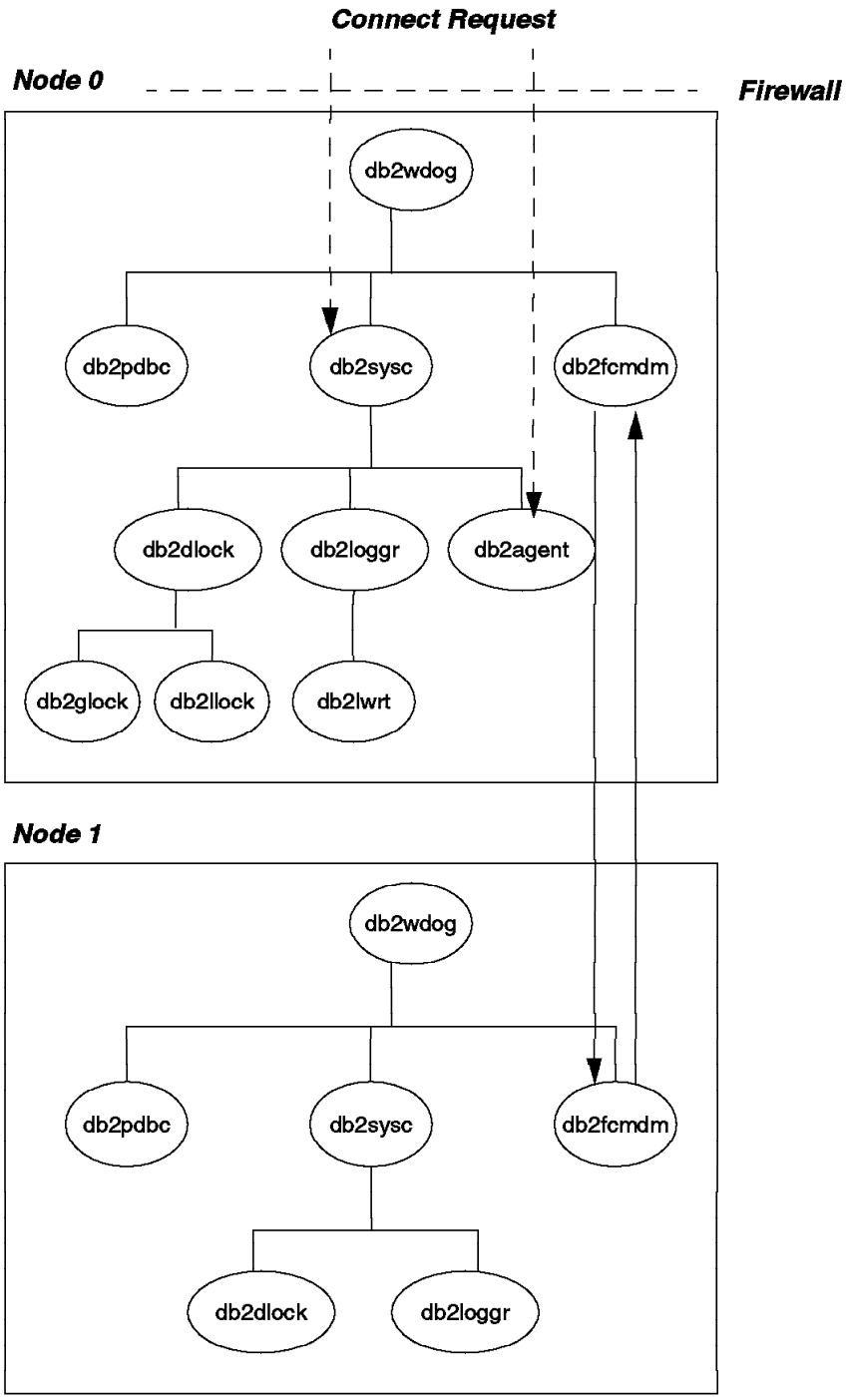


Figure 21. Parent-Child Processes Relationship During Database Connect

After the database manager is started, the database is created, and the database connection is established, the system is ready to process SQL requests. The following major process are created on the catalog node, on the coordinating node, and on other parallel database nodes which are needed to process the parallel requests:

- Parallel agent (db2agntp)

This process handles all the parallel database requests from the coordinator agent (db2agent) or other parallel agents. A pool of the parallel agents can be created during the startup of the database manager. The number of initial parallel agents in the agent pool is controlled by the NUM_INITAGENTS database manager configuration parameter. The maximum size of the pool is controlled by the NUM_POOLAGENTS database manager configuration parameter.

- Local deadlock detector (db2llock)

  This process handles deadlock detection for the local node.

- Child database logger (db2lwrt)

  This process handles logging I/O.

Figure 22 on page 42 illustrates the parent-child processes relationship which is created during a `select` request. In this example, node 0 is the catalog node and the coordinator node. The `select` command is processed on node 0 and node 1.

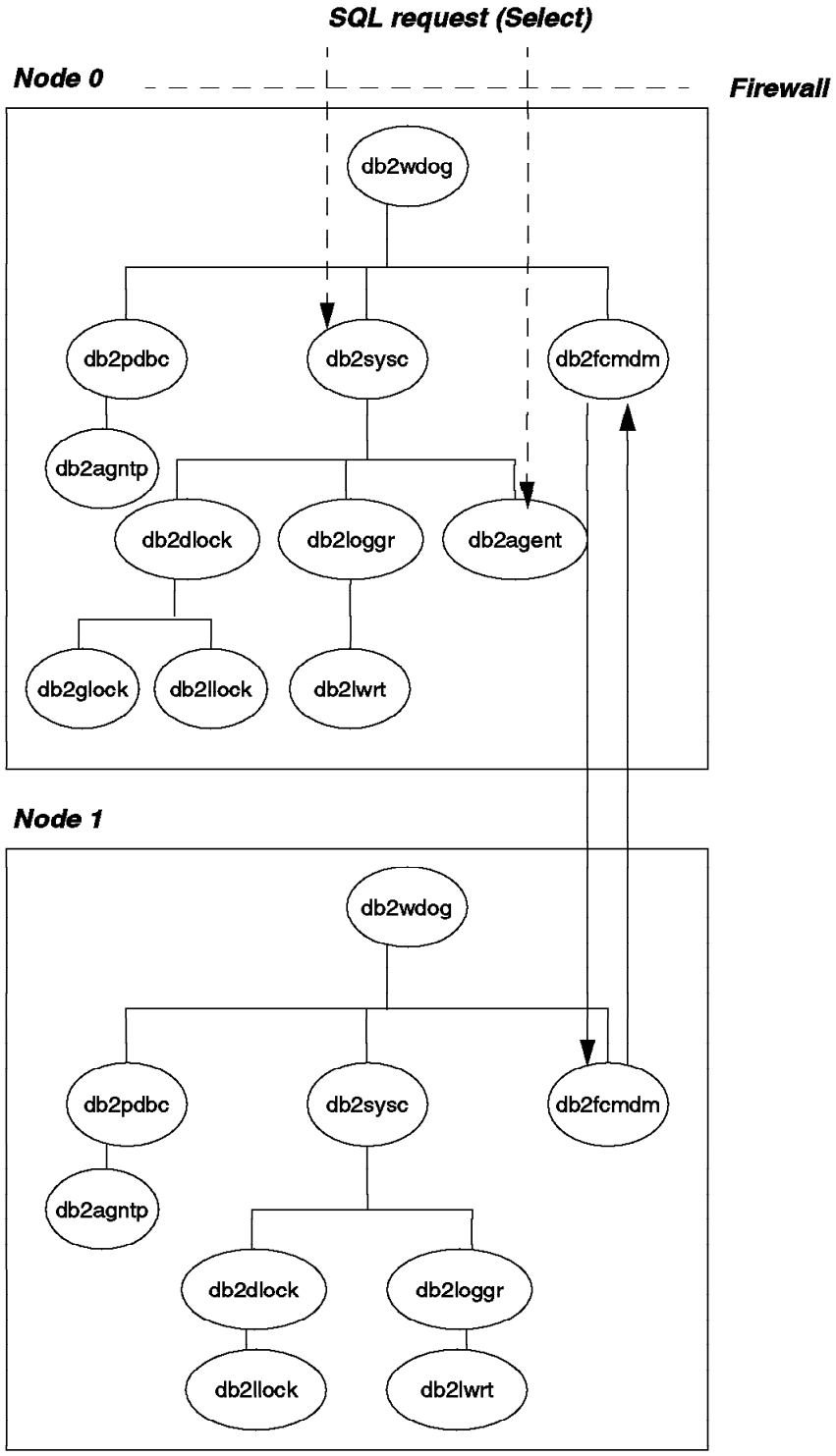


Figure 22. Parent-Child Processes Relationship During Select

After all SQL requests are completed, and the user issues the `connect reset` command, the following major processes are released or removed:

- Local deadlock detector (db2llock)

  This process handles deadlock detection for the local node.

- Child database logger (db2lwrt)

  This process handles logging I/O.

Figure 23 illustrates the parent-child processes relationship which are released or removed during a database connect reset.

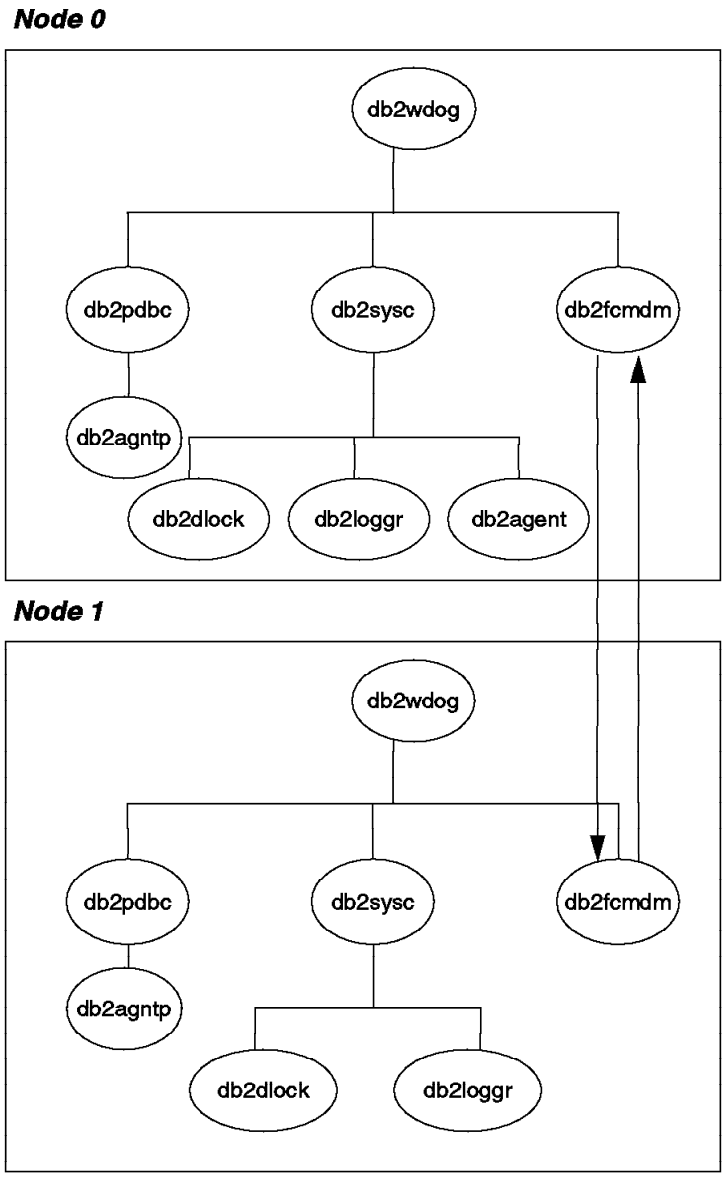


Figure 23. Parent-Child Processes Relationship During Database Connect Reset

3.2.2.1 Control Flow in the Parallel Process Model

The following shows the control flow in the parallel process model during a database request made by an application:

1. At database connect, a coordinating agent (db2agent) is forked by the DB2 system controller (db2sysc) to coordinate the distribution of parallel requests

to other nodes. The node to which the application connects is referred to as the coordinating node.

2. The coordinating agent (db2agent) sends the request from the application to the fast communication manager (db2fcmdm).
3. The fast communication manager (db2fcmdm) forwards the request either to the parallel request queue or directly to the parallel agent request queue for the local node. For a request requiring parallel processing, db2fcmdm communicates with another db2fcmdm across nodes to distribute the parallel requests.
4. If the request is forwarded to the parallel request queue, the parallel system controller (db2pdbc) needs to determine whether to wake up a parallel agent (db2agntp) from the pool or to fork a new one.
5. After the request is completed, the parallel agents (db2agntp) from each node send their replies to their fast communication manager (db2fcmdm). The db2fcmdm of each node sends the replies to the coordinating agent (db2agent).
6. The coordinating agent (db2agent) processes the result and sends it back to the user.

Figure 24 on page 45 illustrates the control flow.

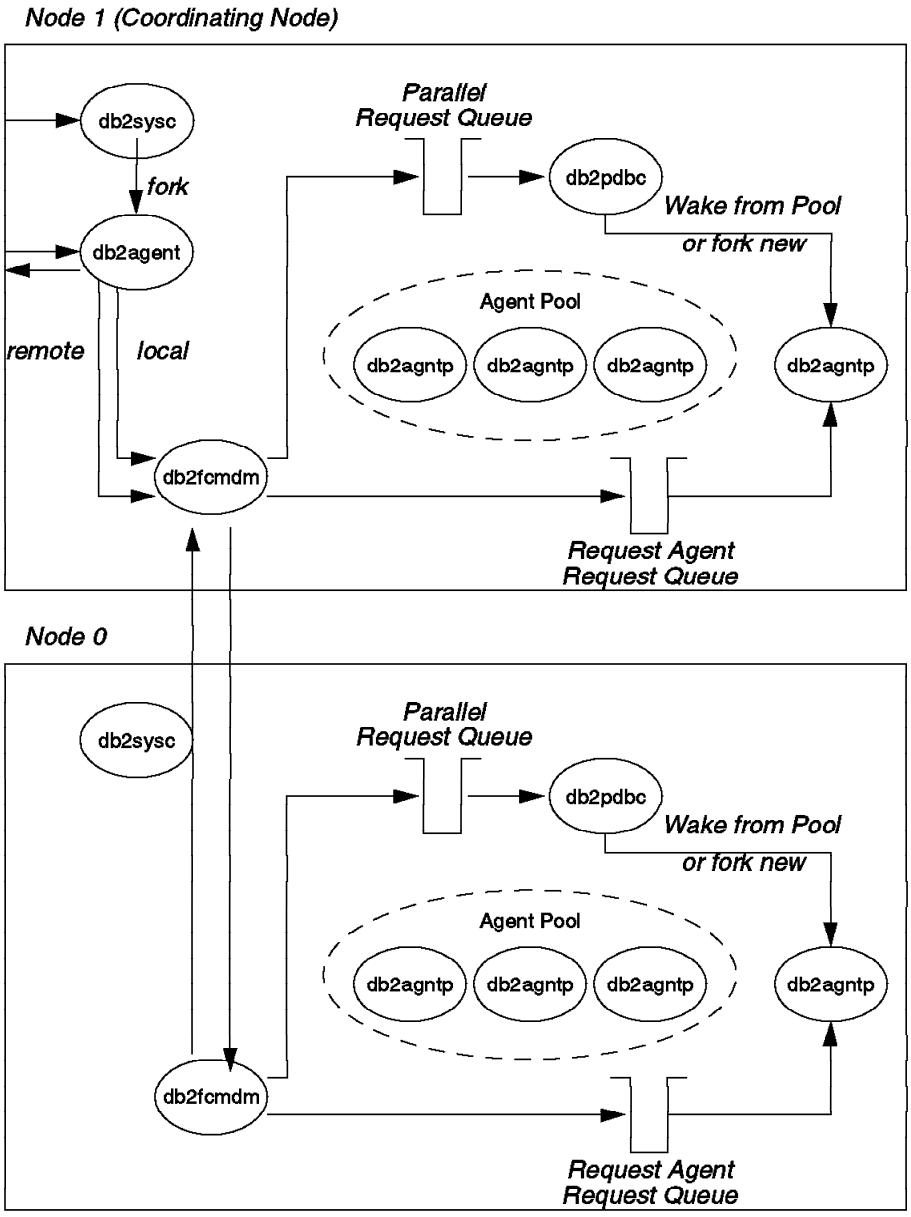


Figure 24. Control Flow in the Parallel Edition Process Model

3.3 Parallel Database Nodes

The mapping of logical database nodes to physical machines is accomplished by the node configuration file, `db2nodes.cfg`. You must create the `db2nodes.cfg` file in the `$HOME/sqllib` directory of the instance. The file contains one entry for each database node that belongs to the DB2 Parallel Edition instance. The file is shared by all database nodes for that instance. Each entry in the file has the format shown in Figure 25 on page 46.

```
node-number    hostname  logical-port netname
```

Figure 25. Node Configuration File, db2nodes.cfg

To enable inter-nodal communication, you must reserve a port number in the /etc/services file. The service name must be DB2_*instance*. (Substitute the instance owner's AIX login name for *instance*.) You may also reserve a range of port numbers in the /etc/services file. If reserving a range of numbers, two service names, DB2_*instance*, and DB2_*instance*_END, are used to mark the beginning and the end of that range. If reserving a range, you must ensure that all the ports in between are available for the DB2 Parallel Edition fast communication manager. The range of ports is recommended to start above 5000.

During database initialization, the fast communication manager (FCM) daemon reads the node configuration file. The logical-port is used as an offset from the start of the range of port numbers in the /etc/services file for the instance. The (hostname, logical-port) pair is used as a well-known address, so it must be unique among all applications.

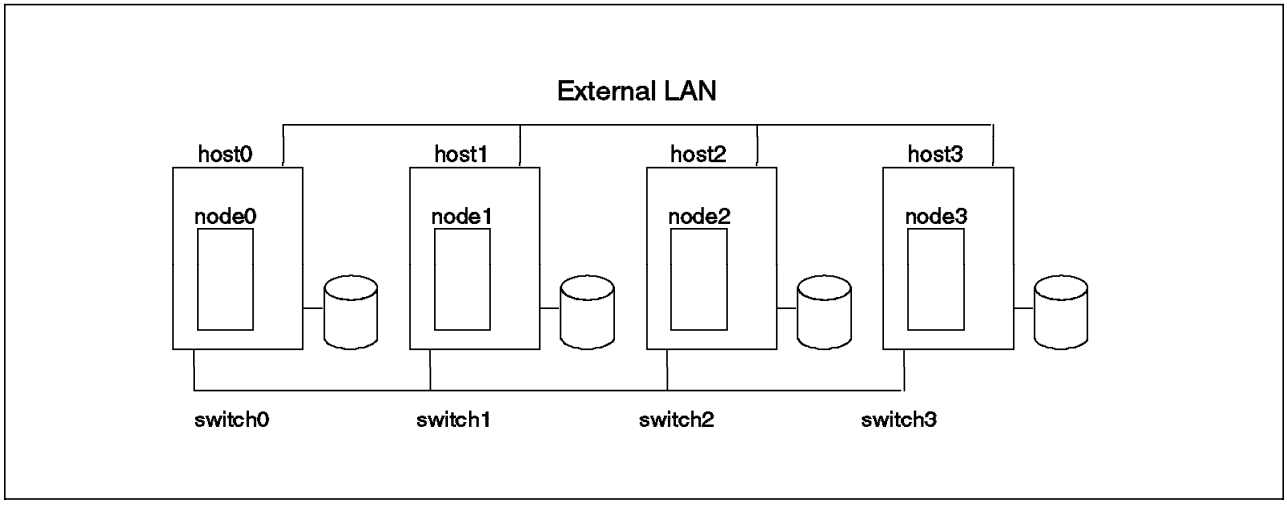


Figure 26. Four Physical Nodes Using the SP Switch

Figure 26 shows an example configuration with four parallel database nodes each of which is mapped to a separate physical machine on a multi-homed host, such as an RS/6000 SP.

Assuming the instance name is *sales*, the /etc/services for the example of Figure 26 is as follows:

```
DB2_sales      5500/tcp
```

The $HOME/sqllib/db2nodes.cfg file for Figure 26 is as follows:

```
0   host0    0   switch0
1   host1    0   switch1
2   host2    0   switch2
3   host3    0   switch3
```

It is possible to have multiple logical parallel database nodes on one physical machine.

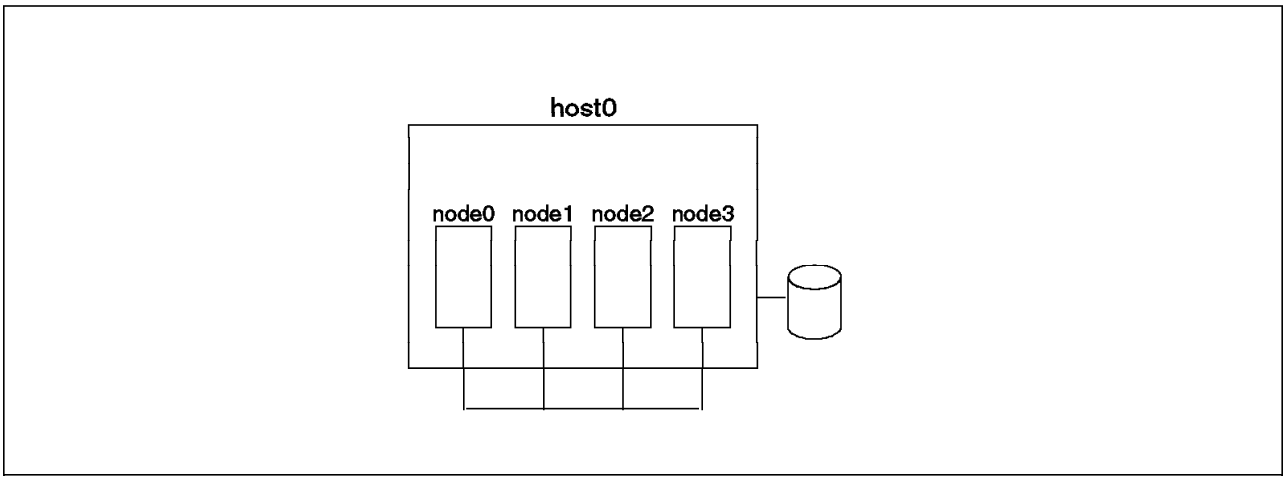


Figure 27. Four Logical Nodes in an SMP Configuration

In Figure 27, four nodes reside on one physical machine. This type of configuration could be used on an SMP machine. An SMP machine has a cluster of individual processors interconnected by shared memory.

Assuming the instance name is *sales*, the /etc/services file for Figure 27 is as follows:

```
DB2_sales       5500/tcp
DB2_sales_END   5503/tcp
```

The $HOME/sqllib/db2nodes.cfg file for Figure 27 is as follows:

```
0   host0    0
1   host0    1
2   host0    2
3   host0    3
```

3.4 Database Instance

A DB2 Parallel Edition instance is defined as a logical database manager environment. Every DB2 Parallel Edition instance is owned by an instance owner, and is distinct from other instances. The AIX login name of the instance owner is also the name of the DB2 Parallel Edition instance. The instance owner must be unique for each DB2 Parallel Edition instance. Each instance can manage multiple databases; however, a single database can only belong to one instance. It is possible to have multiple DB2 Parallel Edition instances on the same group of machines. There are several reasons why you may wish to generate more than one instance on a group of machines.

- To maintain distinct test and production environments
- To use different software levels
- To keep separate SYSADM access on different databases

- To keep separate sets of database manager configuration parameters for performance considerations

Each instance is related to an AIX user, the instance owner. The instance owner owns the files of the instance and has SYSADM authority over the databases belonging to that instance.

3.4.1 Instance Owner

The instance owner has complete control over the instance. The instance owner can start or stop the database manager, modify the database manager configuration, or modify parameters specific to a database.

There is a one-to-one correspondence between an instance and the ownership of the instance. An AIX user cannot be the owner of more than one instance, and an instance must be owned by one AIX user. Any user placed in the same AIX group as the instance owner's primary group will have SYSADM authority over the databases belonging to that instance. There will more than likely be a one-to-one correspondence between the instance owner and SYSADM group.

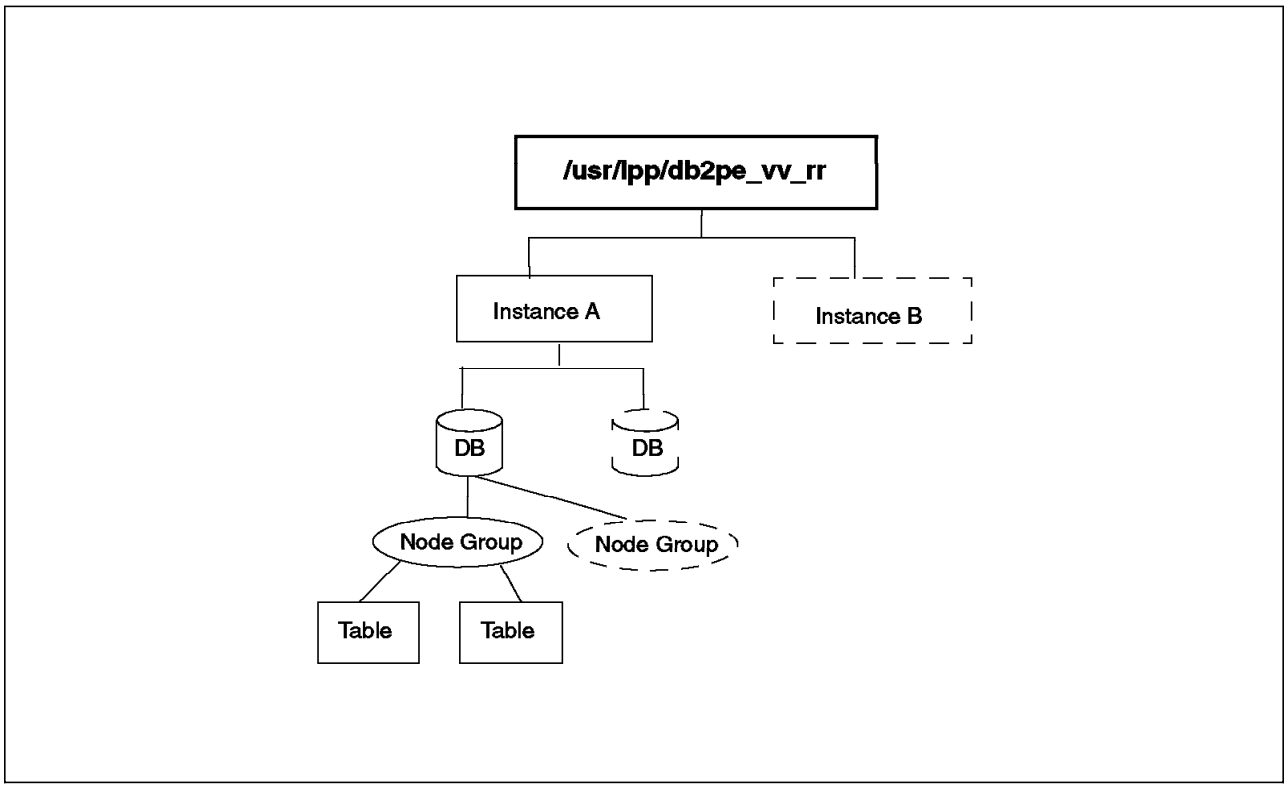


Figure 28. DB2 Parallel Edition Instance

Figure 28 shows two instances, Instance A and Instance B, held on the same PDB nodes or systems. The DB2 Parallel Edition Licensed Program Product (LPP) is installed in the `/usr/lpp/db2pe_01_02` directory. The LPP may be installed either on every node or NFS-mounted to all nodes. The letters vv represent the version of DB2 Parallel Edition; rr is the release. For example, `/usr/lpp/db2pe_01_02` is DB2 Parallel Edition Version 1.2.

Figure 28 also shows the relationship between databases, nodegroups and tables belonging to the same DB2 Parallel Edition instance.

3.5 Database Creation

After the DB2 Parallel Edition instance is created, and the DB2 database manager for all the nodes configured in the instance is active, a database can be created. The database can be created from any node that is configured in the DB2 Parallel Edition instance. The database will be created across all the nodes in the instance. The node on which the `create database` command is invoked becomes the catalog node. All DB2 system catalog tables are stored on the catalog node. Figure 29 illustrates the files generated after the creation of a database.

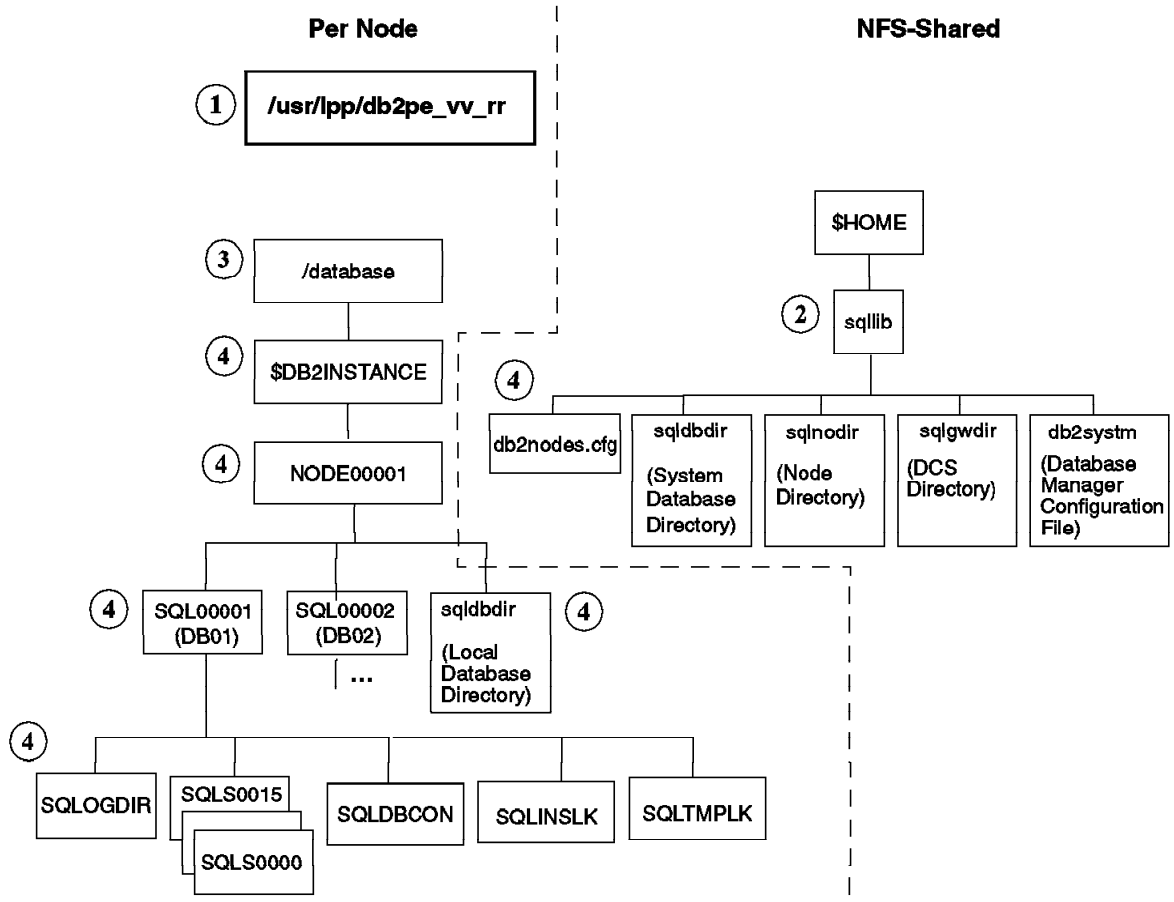


Figure 29. Physical Storage of Parallel Edition Objects

An example of the installation and configuration procedure is found in Chapter 6, "Installation and Configuration" on page 227. The dotted line represented in Figure 29 shows those items that are created on every node and those that are shared between nodes. The numbers in Figure 29 show the physical placement of files and directories when an instance and database are created:

1. The LPP is either installed on all the nodes or shared between nodes.
2. The instance is created on all nodes. The command to create an instance is executed from one of the PDB nodes. It creates the `sqllib` directory of the instance owner.

   When the instance is created, the directory `$HOME/sqllib` is created. The home directory of the instance owner is `$HOME`. This directory must be NFS-mounted to all nodes within the PDB instance. Binary executables, shell

scripts and include files are located in the $HOME/sqllib directory. No other files or directories, other than those created by the DB2 Parallel Edition products, should be placed here in order to avoid the potential loss of data if an instance is deleted.

3. A file system is created, for example /database. We recommend that each node have a file system with the same name and ownership to hold the database. Planning should be done ahead of time to determine such issues as size, potential growth on every node, placement, and file system mirroring. The directories under this file system should be owned by the instance owner and SYSADM group.

4. When the create database command is issued from the node corresponding to node number 1 in the node configuration file, a subdirectory, $DB2INSTANCE/NODE00001, is created in the /database directory. $DB2INSTANCE is the instance owner. Node number 1, in this case, becomes the catalog node of the database. In our example, the following command was issued:

   ```
   db2 create database dss on /database
   ```

   This created the subdirectories, $DB2INSTANCE/NODE00001/SQL00001. SQL00001 is the internal representation of our database, dss. It also places the local database directory or volume directory in the $DB2INSTANCE/NODE00001 directory. In the SQL00001 directory, a number of subdirectories are created:

   - SQLS0000 through SQLS0015 are the 16 default segment directories created. A different number can be created by specifying a different value for the NUMSEGS parameter in the create database command. This value cannot be easily changed after the database is created. For more discussion on NUMSEGS, refer to 3.6, “Database Management and AIX” on page 51.
   - SQLDBCON is the database configuration file. This file cannot be edited directly. To make changes to the database configuration, use the update database configuration command or API.
   - SQLINSLK, and SQLTMPLK are lock files used to ensure that a database is only used by one instance.
   - SQLOGDIR holds the log files for the database. This should be kept in a separate file system on a disk(s) other than where the database is stored to protect against loss of data. It is highly recommended that the logical volume for the log files be mirrored.

   Also shown is the system database directory, sqldbdir. All information about cataloged databases is stored in the system database directory which resides in the $HOME/sqllib directory of the instance owner. Each system database directory entry contains the database name, the database type, whether the database is local or remote, where it is stored (if local), the node name (if remote), and other system information.

   If a remote database is cataloged, there will be a node directory, sqlnodir. The node directory contains entries for all nodes that a client node can access. A DB2 Parallel Edition client can also connect to a remote DB2 Parallel Edition or DB2/6000 database.

   The DCS directory, sqlgwdir is only created if the Distributed Database Connection Services/6000 (DDCS/6000) product is installed, and an entry is made into it. It stores information used by the database manager to access remote databases on host computers. It contains items such as the database name, database alias and the application that prepares requests before sending them to the host.

The db2systm file is the database manager configuration file for the instance. This file is mounted on a shared file system so that all nodes have access to the same file.

The db2nodes.cfg file contains the node configuration for the instance. It maps the parallel database nodes to the physical machines and logical ports. For more information on node configuration file see 3.3, “Parallel Database Nodes” on page 45.

3.6 Database Management and AIX

When DB2 Parallel Edition creates a database, all of the files associated with the database, which are data files, log files and so on, are stored under a directory structure which is, in turn, mounted on a file system. This file system may be configured to be stored on one or more disks. Here, we will discuss the various options the user has for storing data and log files, and the reasons for doing so.

It is advisable to place the log files of a database onto a physical volume which does not contain the data of the database. If a physical volume which is being used to store both data and log files fails, the database would have to be restored from the last offline backup. Because the log files are not available, any changes made since the last offline backup would be lost.

Since the log files are crucial to the recovery of the database, it is advisable to store them on a mirrored logical volume apart from the database data. AIX provides the facility to have up to two mirrored copies of a logical volume.

Data should be distributed across multiple disks. This will enable concurrent access across many disk heads and can improve the overall performance of the system.

There is a maximum file system size restriction of 2 GB in AIX versions up to, and including, Version 3.2.5. For AIX Version 4.1, the maximum size of a file system is 64 GB. However, any individual file in AIX Version 4.1 still has a 2 GB size limitation. Without spreading the physical data of the database over different file systems, the maximum size of a database would be restricted to either 2 GB or 64 GB per node, depending on the level of the AIX operating system. However, DB2 PE allows you to split the database across many file systems over many nodes. The result is that the maximum size of the database becomes the maximum file system size multiplied by the number of file systems over which the database is split, multiplied by the number of nodes.

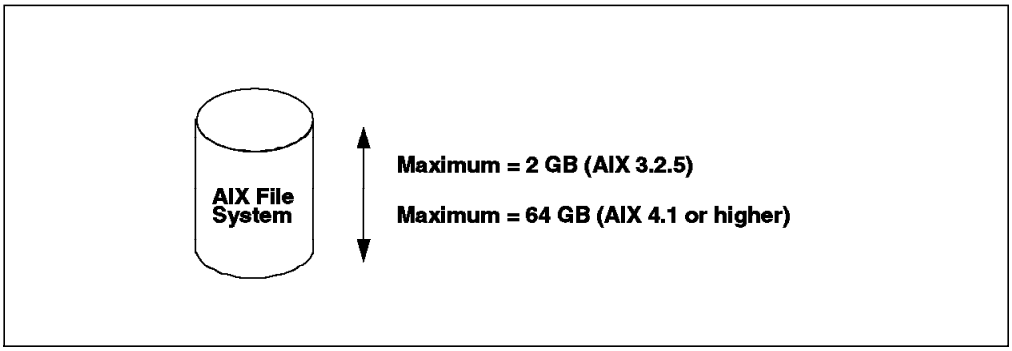


Figure 30. AIX File System Limitation

3.6.1 Mapping Tables to AIX Files

Each table in DB2 PE is given a corresponding AIX file name, for example, SQL00019.DAT. The AIX files are stored in directories. Thus, you have the ability to spread the contents of one table, for example SQL00019.DAT, over many directories. These directories can then be mounted on different AIX file systems.

Mounting is an AIX term referring to connecting a directory mount point to a file system. A file system is a hierarchical structure (file tree) of files and directories.

It is important to make the distinction here between:

- A segment directory, which holds segments (or parts) of a table
- A segment, which is a part of a table

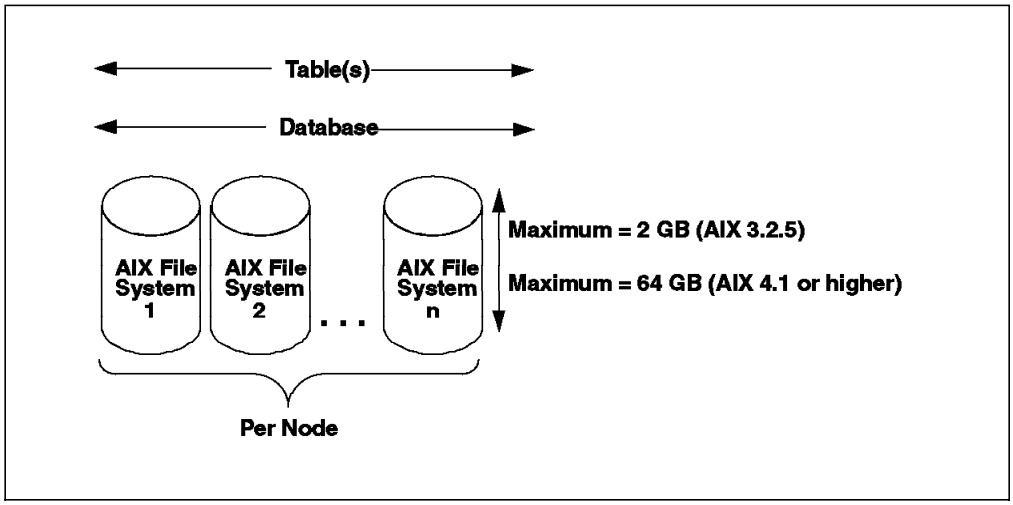


Figure 31. Database Spread Across File Systems

3.6.2 Segment Manager Tool

It is important to understand:

- The database manager is responsible for data in tables and databases.
- The segment manager is a tool that helps you create and manage file systems for database segments.
- The AIX logical volume manager is responsible for the storage of files in the AIX operating system.

Figure 32 on page 53 shows the relationship between the database manager, Segment Manager Tool and the AIX Logical Volume Manager.

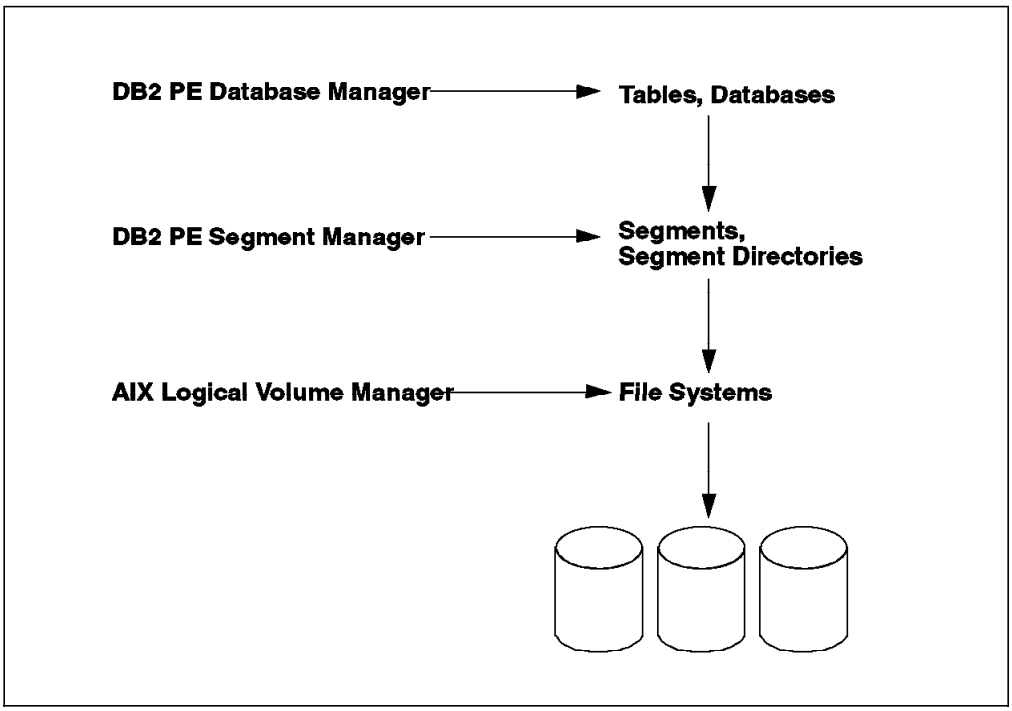


Figure 32. Relationship Between DB2 and AIX

The database manager will only spread the database across multiple directories. If nothing else were done, these directories would still be in one file system, restricted to the AIX file system limit; therefore, the database would be restricted to the same limit.

There is a process to mount file systems over these segment directories. There is a tool provided to do this: the DB2 Segment Manager Tool. The Segment Manager Tool is described in more detail in 5.2, “Segment Manager Tool” on page 142.

3.6.3 Multi-Page File Allocation

In DB2 Parallel Edition V1.1, the page allocation for database files such as .DAT, .INX, .LF, and .TDA is by one 4 KB page at a time when a new page is needed.

In V1.2, the page allocation has been enhanced to reduce the overhead so the performance for data insertion and index creation can be improved. When a new page is needed, the database manager will expand the database files by *N* pages at a time, where *N* is the SEGPAGES value defined during database creation.

Note: This multi-page file allocation only applies to .DAT, and .INX files, not .LF, or .TDA files. Also it does not apply to the first *N* pages of .DAT and .INX files. This is done to ensure that over allocation will not happen for small tables.

DB2 Parallel Edition V1.2 supports both ways of expanding the database files.

- Databases created before V1.2 will continue using the page allocation logic in DB2 Parallel Edition V1.1, meaning that the database file will be expanded one page at a time. A tool is provided to enable the multi-page file allocation logic. It is described in more detail in 3.6.3.1, “Enabling Multi-Page File Allocation Logic” on page 54.

- Databases created in V1.2 will use the multi-page file allocation logic for .DAT and .INX files. The page allocation for .LF and .TDA files, and the first SEGPAGES pages of .DAT and .INX files will still be one page at a time.

3.6.3.1 Enabling Multi-Page File Allocation Logic

A new parameter, multi_page_alloc, in the database configuration is introduced in V1.2 to indicate if the multi-page file allocation logic is enabled. The value can be:

- YES to indicate the method for expanding a file is by SEGPAGES pages. Remember the SEGPAGES pages will still be allocated one at a time.
- NO to indicate the method for allocating a file is by one page.

Note: The multi_page_alloc parameter cannot be modified by the update database configuration command.

The tool db2empfa is used to change the value for multi_page_alloc to YES. It resides in the instance owner's home directory, $HOME/sqllib/bin.

The syntax for this tool is as follows:

```
db2empfa database-alias
```

The following considerations apply when running db2emfa:

- db2empfa has to be executed on all the nodes where the database is partitioned.
- It requires an exclusive database connection.
- It cannot be executed concurrently on a catalog node and a non-catalog node. You must execute the db2empfa utility on a catalog node by itself, and then execute it on the non-catalog nodes.

Once the db2empfa has been executed on the catalog node the following command can be used to executed the db2empfa utility on all other nodes in parallel:

```
db2_all ";<<-3< db2empfa database-alias"
```

For the number three (3) you would substitute the appropiate node that contains the system catalog.

Note: Once the value for multi_page_alloc parameter is set to YES, it is not possible to disable the multi-page file allocation logic without restoring from a database backup taken before the multi_page_alloc was set.

3.6.4 Default Database Settings

The default settings for creating a database are:

- There are 16 segment directories.
- When a table reaches 32 pages in size, it will overflow into the next segment directory.

Given these default values, the maximum size of a database in AIX 3.2.5 would be:

```
Maximum database size = ( default(NUMSEGS) * max(file system size)
                      = (       16         *        2 ) GB
                      = 32 GB (AIX 3.2.5)
```

> Using the defaults, the maximum size of a database in AIX 3.2.5 is 32 GB per node.

Given these default values, the maximum size of a database in AIX 4.1 would be:

```
Maximum database size = ( default(NUMSEGS) * max(file system size)
                      = (       16         *       64 ) GB
                      = 1024 GB (AIX 4.1)
```

> Using the defaults, the maximum size of a database in AIX 4.1 is 1024 GB per node.

There are two parameters which control the segmentation of databases:

- NUMSEGS, which is the number of segment directories over which the database is spread
- SEGPAGES, which is the number of 4 KB pages in a segment

These parameters are set at database creation and can only be changed by dropping, and recreating, the database.

Use the Command Line Processor (CLP) or the command line to specify these values. For example, from the command line, the syntax would be:

```
db2 create database asample NUMSEGS 10 SEGPAGES 64
```

3.6.5 Storage of Objects in Segment Directories

The following formula defines the segment directory which stores the first page of a database file.

```
Segment Directory Number = MODULO(FID/NUMSEGS)
```

where FID (File ID) can be found by:

```
db2 " select name,creator,fid from sysibm.systables where type = 'T' "
```

FID is a unique identifier for a table used by the database manager. When a new table is created, it is allocated the next free FID.

For example:

```
NAME                CREATOR  FID
------------------- -------- ------
SYSTABLES           SYSIBM        2
SYSCOLUMNS          SYSIBM        3
SYSINDEXES          SYSIBM        4
SYSVIEWS            SYSIBM        5
SYSVIEWDEP          SYSIBM        6
SYSPLAN             SYSIBM        7
SYSPLANDEP          SYSIBM        8
SYSSECTION          SYSIBM        9
SYSSTMT             SYSIBM       10
SYSDBAUTH           SYSIBM       11
SYSPLANAUTH         SYSIBM       12
SYSTABAUTH          SYSIBM       13
SYSINDEXAUTH        SYSIBM       14
SYSRELS             SYSIBM       15
SYSNODEGROUPS       SYSIBM       16
SYSPARTITIONMAPS    SYSIBM       17
SYSNODEGROUPDEF     SYSIBM       18

  17 record(s) selected.
```

So, for a user table whose FID = 19 and, given the default, NUMSEGS = 16:

```
Segment Directory Number = MODULO(FID/NUMSEGS)
Segment Directory number = 19 modulus 16
Segment Directory number = 3 which is SQLS0003
```

The MODULO 16 part of the equation is the remainder after dividing by 16. FIDs of 0 and 1 are reserved in DB2 Parallel Edition. If FID=19, the segment directory is SQLS0003. When FID=20, the segment directory is SQLS0004. For example:

```
FIDs             Seg Dir #
---              ---------
 N/A  16 ......... 0
 N/A  17 ......... 1
  2   18 ......... 2
  3   19 ......... 3
  4   20 ......... 4
  5   21 ......... 5
  6   22 ......... 6
  7   23 ......... 7
  8   24 ......... 8
  9   and so on.    9
 10   ............ 10
 11   ............ 11
 12   ............ 12
 13   ............ 13
 14   ............ 14
 15   ............ 15
```

Given an initial segment directory number of 3, the first .DAT file for the user table data will be stored in a directory called SQLS0003 under the database directory. Should the size of this AIX file exceed 32 pages, another .DAT file with the same name will be created in the SQLS0004 directory, and subsequent inserts will be written to this second file. As the table grows, the .DAT files will be placed in subsequent segment directories in order. If the last directory written

into is SQLS000n, with n being the upper limit, writing will continue in the SQLS0000 directory.

3.6.6 Determining the Values for NUMSEGS and SEGPAGES

Care should be taken in determining the values of NUMSEGS and SEGPAGES because once a database is created, these values cannot be easily changed. You must back up the database, drop it and then re-create it with different values for NUMSEGS and SEGPAGES. Since changing the segmentation of your database is not a trivial task after the database is created, it is important to accurately plan for your particular requirements.

Fortunately, in most cases, the default value for SEGPAGES is adequate to give you:

- Flexibility in case your database grows
- Small performance overhead (except in actual database creation time) in having more segment directories than is necessary at database creation

Begin by estimating the size of your tables, databases and temporary tables. For more information, see the *DB2/6000 Administration Guide*. After determining your table and database size, you should be able to estimate the value of NUMSEGS to allow for future growth.

When determining SEGPAGES, keep in mind that this is the number of 4 KB pages that will be placed in every segment directory. In DB2 Parallel Edition V1.2, when multi-page file allocation logic is enabled, files will be allocated SEGPAGES pages at a time, except for the first SEGPAGES pages. To avoid having unnecessary amounts of space allocated in the segment directories, selecting a large value for SEGPAGES is not advisable. The default of 32 pages should work for most environments. Figure 33 shows a potential problem when underestimating the value of NUMSEGS and/or selecting too large a value for SEGPAGES.

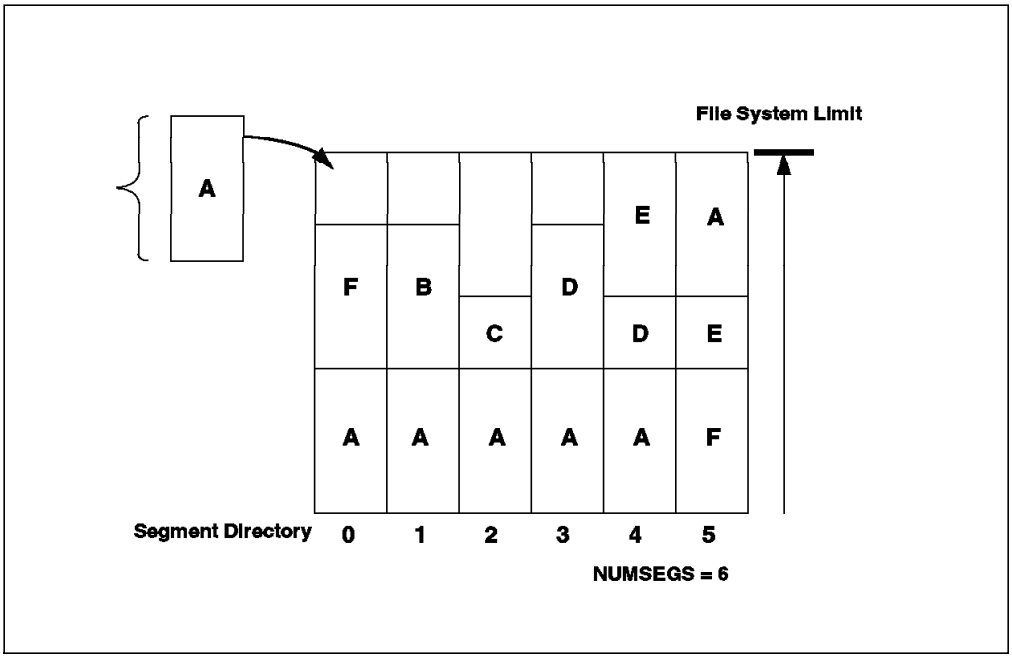


Figure 33. Segmented Tables Example

Figure 33 is a database that has NUMSEGS=6. A data table, indicated by segment A, has an FID of 32. This segment A is represented in DB2 PE as a .DAT file, for example, SQL00032.DAT. The first segment is placed in segment directory 0. The second in segment directory 1, and so on. The problem is that when the table has placed a segment in every segment directory, the database manager will try to allocate more space in segment directory 0. If segment directory 0, does not have enough free space, you will not be able to write into that segment directory. Even if space were to remain in other segment directories associated with the database, you would receive an SQL error code (SQL0968C) indicating that the file system was full.

3.6.7 Maximum Size for Tables and Databases

The maximum size of a database is dependent on the maximum values NUMSEGS, and SEGPAGES may take and the number of nodes in your configuration. A page is 4096 bytes or 4 KB.

SEGPAGES may take the following values:

Minimum	4 pages
Default	32 pages
Maximum	524288 pages (2 GB)

Notice that the size of any one file or segment is 2 GB on both AIX 3.2.5 or AIX 4.1.

NUMSEGS may take the following values:

Minimum	1
Default	16
Maximum	256

Maximum table capacity is an architectural limit that is defined by:

```
Maximum Table Size = 64 GB Per Node
```

Maximum database capacity for AIX 3.2.5 is defined by:

```
Max database size = ( max(NUMSEGS) * max(file system size) )
                  = (     256      *      2 ) GB (AIX 3.2.5)
                  = 512 GB Per Node
```

```
Maximum Database Size = 512 GB Per Node
```

Maximum database capacity for AIX 4.1 is defined by:

```
Max database size = ( max(NUMSEGS) * max(file system size) )
                  = (     256      *      64 ) GB (AIX 4.1)
                  = 16384 GB or 16 TB Per Node
```

```
Maximum Database Size = 16 TB Per Node
```

DB2 PE can support extremely large databases. However, the management and maintenance of databases of large proportions needs careful consideration.

3.6.8 Performance and Resource Considerations

The following are factors that you may want to consider for your environment:

- Setting SEGPAGES small would help distribute data among different segment directories, hence, different file systems. This would help balance the writing of data among the file systems. One consideration for selecting too small a size for SEGPAGES is how it might affect the MAXFILOP database parameter. The database parameter, MAXFILOP, allows you to limit the number of database files open at any one time for one database manager agent. If opening a file causes this limit to be exceeded, a database file that is open and in use by this application is closed. If MAXFILOP is too small, the overhead of opening and closing files so as not to exceed this value may degrade performance. This information can be retrieved using the Database System Monitor:

  ```
  db2 get snapshot for database on asample | grep closed

  Database files closed                       = 0
  ```

  However, setting this value large to balance a small SEGPAGES size will affect system resources, such as memory. The recommendation is to leave SEGPAGES at either 32 or 64 pages. There would be no benefit to setting SEGPAGES larger than 256.

  Note: Setting the SEGPAGES database parameter to large values is not advisable when the multi-page file allocation logic is enabled, to avoid overallocation of space in the segment directories.

- The larger the database will be, the higher NUMSEGS must be. The higher NUMSEGS, the more segment file systems to manage.
- A guideline for NUMSEGS is to set it to the number of disks attached to any one node. If there are different numbers of disks attached to each node, select the smallest number in the configuration.

3.6.9 Changing the Maximum Size of a Database

The value of NUMSEGS, or SEGPAGES cannot be changed after creating a database. The only way to change these values and retain your data is to:

- Back up (or export) the database
- Drop the database
- Create the database with different NUMSEGS/SEGPAGES values
- Restore (or import/load) the data into this new database

Obviously, for large databases on the gigabyte or even terabyte scale, this is a time-consuming process.

3.7 Nodegroups and Data Partitioning

In DB2 Parallel Edition, data placement is one of the more challenging tasks. It determines the best placement strategy for all tables defined in a parallel database system. The rows of a table can be distributed across all the nodes (fully declustered), or a subset of the nodes (partially declustered). Tables can be assigned to a particular set of nodes. Two tables in a parallel database

system can share exactly the same set of nodes (fully overlapped), at least one node (partially overlapped), or no common nodes (nonoverlapped). Parallel Edition supports the hash partitioning technique to assign a row of a table to a table partition.

3.7.1 Nodegroups

Nodegroups in DB2 Parallel Edition are used to support declustering (full or partial), overlapped assignment and hash partitioning of tables in the parallel database system. A nodegroup is a named subset of one or more of the nodes defined in the node configuration file, `$HOME/sqllib/db2nodes.cfg`, of DB2 Parallel Edition. It is used to manage the distribution of table partitions. In DB2 Parallel Edition, there are system-defined and user-defined nodegroups. Tables must be created within a nodegroup.

Figure 34 shows a DB2 Parallel Edition instance consisting of four physical machines. The default nodegroup, `IBMDEFAULTGROUP`, spans all of the nodes in the instance. The catalog nodegroup, `IBMCATGROUP`, is created only on the node where the `create database` command was issued. The catalog node is on host0 in this example. Also shown is a user-defined group that spans host2 and host3.

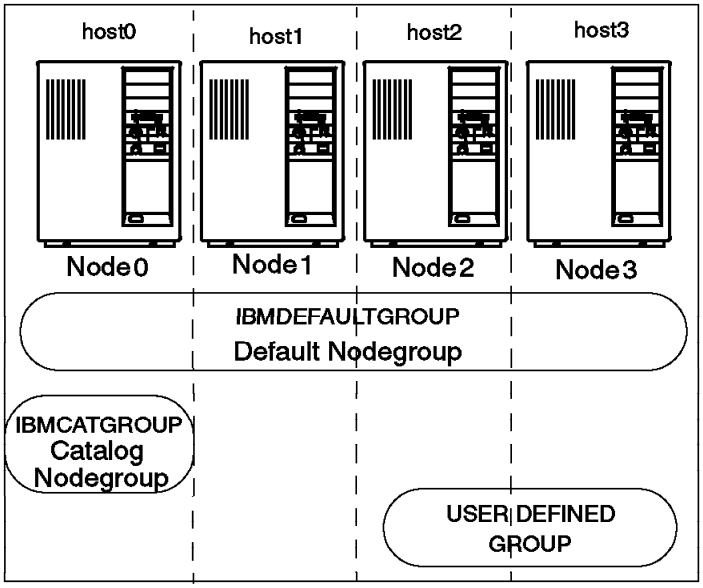


Figure 34. Nodes and Nodegroups

Two system-defined nodegroups are automatically generated by the Parallel Edition database manager at database creation time. They are:

1. `IBMCATGROUP` is a single-node nodegroup which contains only the catalog node. The catalog node is any node where the `create database` command was issued. It cannot be altered by using the `redistribute nodegroup` command operation and is protected from the `drop nodegroup` SQL statement.
2. `IBMDEFAULTGROUP` contains all the nodes defined in the node configuration file ($HOME/sqllib/db2nodes.cfg) of DB2 Parallel Edition at database creation time. It is the default nodegroup for the `create table` SQL statement. It can

be altered by using the redistribute nodegroup command to add or remove a node. It is protected from the drop nodegroup SQL statement.

A user-defined nodegroup can be created by using create nodegroup SQL statement. You can specify the set of nodes for your table partition. This nodegroup can be altered by using the redistribute nodegroup command to add and remove nodes. It can be dropped by using the drop nodegroup SQL statement.

3.7.2 Creating a Nodegroup

When the create nodegroup SQL statement executes, the DB2 Parallel Edition database manager:

- Creates a partitioning map for the nodegroup
- Generates a partitioning map ID
- Inserts records into SYSIBM.SYSNODEGROUPS, SYSIBM.SYSPARTITIONMAPS, and SYSIBM.SYSNODEGROUPDEF

The following example illustrates what happens before and after the execution of the create nodegroup SQL statement. The name of the nodegroup used in this example is called groupa. Nodegroup groupa is defined on a two-node (0 and 1) system. Before the execution of the SQL statement, the parallel database system only has the two system-defined nodegroups, IBMCATGROUP, and IBMDEFAULTGROUP. After the execution of the SQL statement, the user-defined nodegroup, GROUPA, is created, and the entry is inserted into the catalog tables. You can issue the list nodegroups command to display the information about the nodegroups defined to the system.

- The list nodegroups output before execution of the create nodegroup SQL statement:

```
db2 list nodegroups show detail

NAME                 PMAP_ID      NODENUM      IN_USE
-------------------- ------------ ------------ ----------
IBMCATGROUP             0            0            Y
IBMDEFAULTGROUP         1            0            Y
IBMDEFAULTGROUP         1            1            Y
```

- To create the nodegroup for groupa:

```
db2 create nodegroup groupa on nodes(0,1)
```

- The list nodegroups output after the execution of the create nodegroup SQL statement:

```
db2 list nodegroups show detail

NAME                 PMAP_ID      NODENUM      IN_USE
-------------------- ------------ ------------ ----------
IBMCATGROUP             0            0            Y
IBMDEFAULTGROUP         1            0            Y
IBMDEFAULTGROUP         1            1            Y
GROUPA                  2            0            Y
GROUPA                  2            1            Y
```

3.7.3 Considerations for Nodegroups

If you want to split the data across all nodes defined in the node configuration file, `$HOME/sqllib/db2nodes.cfg`, you can create your tables in the default nodegroup, `IBMDEFAULTGROUP`. In some cases, you may want to create your own single-node or multi-node nodegroups to improve performance or to gain faster recoverability.

3.7.3.1 Table Collocation

By far the most important consideration for nodegroups is providing for table collocation. Because the collocated join strategy does not require internodal communication, it provides the best possible query performance.

To have table collocation, you must:

- Place the tables in the same nodegroup.
- Ensure that their respective partitioning keys have the same number of columns, and the corresponding columns are partition compatible.

Until you know exactly what you are doing, put all of your tables in the IBMDEFAULTGROUP nodegroup.

3.7.3.2 Table sizes

Avoid extending medium-sized tables across too many nodes. For example, a 100 MB table may perform better on a 16-node nodegroup than on a 32-node nodegroup. The degree of parallelism achieved by the larger number of nodes needs to be offset by the increased costs with their management. These costs include broadcasting access plan subsections to entire nodegroups, broadcasting table queues to multiple nodes, and receiving sorted table queues from multiple nodes.

For performance reasons, you should keep the volume of raw data per node below 20 GB.

3.7.3.3 Single-Node Nodegroups

If the size of the tables is small, they can be placed in single-node nodegroups. However, if you want to have collocation with a larger table, the small table should be spread across the same set of nodes as the larger table.

In a single-node nodegroup, tables do not require a partitioning key. By default, a table in a single-node nodegroup is created without a partitioning key.

Tables consisting of only long-field (LONG VARCHAR or LONG VARGRAPHIC) columns must be placed in single-node nodegroups because there are no columns on which to define a partitioning key.

If a single-node nodegroup will ever be changed (redistributed) into a multi-node nodegroup, all of the tables in the nodegroup must have a partitioning key. This may dictate separate single-node nodegroups for tables which have, and which do not have, partitioning keys.

3.7.3.4 Fast Recovery

If you require fast recoverability, place user tables in nodegroups that exclude the catalog node. The RESTORE and OFFLINE BACKUP utilities connect to the database in exclusive mode at the node where the utility is running and in share mode at the catalog node. This means that when the utilities are running on the catalog node, the exclusive lock will prevent any other concurrent activity on any of the database nodes. By default the IBMDEFAULTGROUP nodegroup includes the node containing the system catalog tables. The REDISTRIBUTE NODEGROUP command can be used to drop the catalog node from the nodegroup.

3.7.3.5 Dedicated Processor for Catalog Node

If the database will be processing large numbers of dynamic SQL statements, as is typical of decision-support applications, it is recommended to have a dedicated processor to the catalog node in large-system configurations. If this is not possible, the REDISTRIBUTE NODEGROUP command can be used to reduce the number of data partitions that hash to the catalog node.

The access plans for static SQL statements are normally cached at the coordinator nodes, so a dedicated processor is less useful in this environment.

Some access plans require a subsection to execute on a single-node nodegroup. These subsections are used in some forms of view materialization. Because the only single-node nodegroup that exists in all configurations is IBMCATGROUP, the optimizer will use this nodegroup as needed. If this type of access plan is typical in your installation, a dedicated processor for the catalog node may be desirable.

3.7.3.6 Dedicated Coordinator Nodes

In some cases it may be desirable to have a set of nodes that are dedicated to performing coordinator functions.

For example, the large number of Micro Channel slots in RS/6000 SP wide nodes may make them desirable for providing network connectivity to client machines.

The software licensing costs associated with end-user query tools may dictate their installation be restricted to a subset of the nodes. Query tools that do significant amounts of processing may perform best with SMP nodes for coordinators.

Sometimes significant query processing is performed by the coordinator subsection of the parallel access plans. These include global aggregation predicates and ordering columns different from grouping columns.

3.7.3.7 Directed Inner-Table and Outer-Table Joins

Small tables that are used as the inner table of a directed inner-table and outer-table join should be defined on at least as many tables as the outer table. This is because the join will be processed on the nodes composing the nodegroup of the inner table.

Directed inner- and outer-table joins are processed on the nodes composing the nodegroup of the inner table. The inner tables should be defined on sufficient nodes to handle the expected data volumes of both tables. The directed inner table and outer table join strategy is often selected to implement non-collocated outer joins. In this case the optimizer has no choice over the inner table, because right outer joins are rewritten by the optimizer as left outer joins.

Performance would suffer greatly if the inner table were defined in a single-node nodegroup.

3.7.3.8 Miscellaneous Considerations

If two tables are not to be collocated, it may be better to put them in separate nodegroups because the nodegroups can be tuned independently. This will provide for better correction of data skew or different growth rates via customized partitioning maps.

Some end-user query tools that tolerate (as opposed to exploit) DB2 Parallel Edition might not specify a target nodegroup when defining temporary tables. These tables would then be created in the IBMDEFAULTGROUP nodegroup and have a default partitioning key. The IBMDEFAULTGROUP nodegroup must have sufficient disk space and processor nodes to handle the expected volume. It may also be possible to design your nodegroup and partitioning key strategies so these temporary tables are collocated with larger permanent tables.

The `REDISTRIBUTE NODEGROUP` command can be used to tailor data volumes and processing loads to non-symmetric hardware configurations. Keep in mind, however, that the optimizer uses the database configuration parameters of the coordinator node when compiling access plans.

Data access patterns may also be a factor in nodegroup design. Medium-sized tables that are accessed concurrently may perform better in different, non-intersecting nodegroups.

If you have a parallel database system that supports both decision support system (DSS) and online transaction processing (OLTP) users, consider the following:

- If DSS and OLTP users go after different sets of data, place OLTP and DSS data in different nodegroups or even in different databases.
- If DSS and OLTP users go after the same sets of data, try to focus on DSS since OLTP applications are normally static SQL, and access plans are cached at the coordinator nodes.

3.7.4 Data Partitioning

DB2 Parallel Edition supports the hash partitioning technique to assign each row of a table to the node to which the row is hashed. You need to define a partitioning key before applying the hashing algorithm. The hashing algorithm uses the partitioning key as an input to generate a partition number. The partition number then is used as an index into the partitioning map. The partitioning map contains the node number(s) of the nodegroup. There are three major steps to perform the data partitioning:

1. Partitioning key selection
2. Partitioning map creation
3. Partitioning rows

3.7.5 Partitioning Key

A partitioning key is a set of one or more columns of a given table. It is used by the hashing algorithm to determine on which node the row is placed.

The partitioning key is defined by using the `create table` or `alter table` SQL statement. Any columns with a data type other than long field (that is, not LONG VARCHAR or LONG VARGRAPHIC) can be used to form a partitioning key. If the partitioning key is not specified at table creation time, DB2 Parallel Edition will use the following rules to define a default partitioning key:

- First column of a primary key
- First non-LONG field column

Note: The DB2/6000 `create table` SQL statement will work in DB2 Parallel Edition without modification, providing there is at least one non-LONG field column. In this case, the default partitioning key and nodegroup will be used. Tables without a partitioning key are only allowed in single-node nodegroups.

The following is an example of creating a partitioning key while creating a table:

```
CREATE TABLE order (o_orderkey       integer  not null,
                    o_custkey        integer  not null,
                    o_orderstatus    char(1)  not null,
                    o_totalprice     float    not null,
                    o_orderdate      date     not null,
                    o_orderpriority  char(15) not null,
                    o_clerk          char(15) not null,
                    o_shippriority   integer  not null,
                    o_comment        char(49) not null)
IN four
PARTITIONING KEY (o_orderkey) USING HASHING
```

In the example, the partitioning key is defined as the `o_orderkey` column, which will be used to distribute the rows of table order across nodegroup four.

3.7.5.1 Partitioning Key Selection and Considerations

A partitioning key has direct impact on performance; therefore, choosing a good partitioning key is important. In order to choose a good partitioning key, you must understand the nature of the queries and join strategies used in your system. For more detail on join strategies, refer to 4.8, “Join Operations” on page 102.

The following are recommendations for choosing a partitioning key:

- The partitioning key for each table in a nodegroup determines if the tables are collocated. The collocation of tables has a significant impact on performance.
- If collocation is not a major consideration, a good partitioning key for a table is one that spreads the data evenly on all nodes in the nodegroup.
- The cost of applying the partitioning function is proportional to the size of the partitioning key. Unnecessary columns should not be included in the partitioning key.
- The partitioning key should be formed with a minimum number of columns, typically from one to three columns. The fewer partitioning key columns that need to be specified, the more likely the parallel optimizer is able to use collocated or directed join strategies.

- The partitioning key should include the most frequently joined columns.
- The partitioning key should include columns that often participate in a GROUP BY clause.
- Columns with skewed data and columns with a small domain should not be chosen for use as a partitioning key. Columns with a large domain and skewed data, however, can be effective partitioning key if you can correct the skew. The partitioning key should have enough distinct values to ensure an even data distribution, and a scalable configuration.
- The partitioning key should not be frequently updated. Any updates must be done as deletion and insertion operations.
- An integer partitioning key is more efficient than a character key, which is more efficient than a decimal key.

The following restrictions apply to the partitioning key:

- Long-field (LONG VARCHAR or LONG VARGRAPHIC) columns are not allowed to be used as part of the partitioning key.
- Any unique index or primary key must be a superset of the partitioning key.
- Alteration of the partitioning key definition is not allowed for tables in multinode nodegroups.
- The db2split sample program does not support float or graphic partitioning key columns.

3.7.5.2 Partition Compatibility

It is possible to have different column types in a table with the same value. For example, the number 8 can be represented with INTEGER, DECIMAL, and FLOAT data types. Some data types with the same value will map to the same partitioning map number by using the hashing algorithm. These data types are called partition compatible.

In DB2 Parallel Edition, partition compatibility is shown in Table 3 on page 67 with the following comments:

- Partition compatibility is not affected by columns with NOT NULL or FOR BIT DATA definitions.
- NULL values of compatible data types are treated identically. Different results might be produced for NULL values of non-compatible data types.
- Decimals of the same value in the partitioning key are treated identically, even if their scale and precision differ.
- Trailing blanks in character strings (CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC) are ignored by the system-provided hashing function.
- CHAR or VARCHAR of different lengths are compatible data types.

Table 3 on page 67 shows the summary of the compatibility of data types in partitions.

Table 3. Partition Compatibilities

	smallint	integer	decimal	float	char	varchar	graphic	vargraphic	date	time	timestamp
smallint	Yes	Yes	No	No	No	No	No	No	No	No	No
integer	Yes	Yes	No	No	No	No	No	No	No	No	No
decimal	No	No	Yes	No	No	No	No	No	No	No	No
float	No	No	No	Yes	No	No	No	No	No	No	No
char	No	No	No	No	Yes	Yes	No	No	No	No	No
varchar	No	No	No	No	Yes	Yes	No	No	No	No	No
graphic	No	No	No	No	No	No	Yes	Yes	No	No	No
vargraphic	No	No	No	No	No	No	Yes	Yes	No	No	No
date	No	No	No	No	No	No	No	No	Yes	No	No
time	No	No	No	No	No	No	No	No	No	Yes	No
timestamp	No	No	No	No	No	No	No	No	No	No	Yes

3.7.6 Partitioning Map

In DB2 Parallel Edition, a partitioning map is an array of 4096 node numbers. Each nodegroup has a partitioning map. The content of the partitioning map consists of the node numbers which are defined for that nodegroup. The hashing algorithm uses the partitioning key on input to generate a partition number. The partition number has a value between 0 and 4095. It is then used as an index into the partitioning map for the nodegroup. The node number in the partitioning map is used to indicate to which node the operation should be sent.

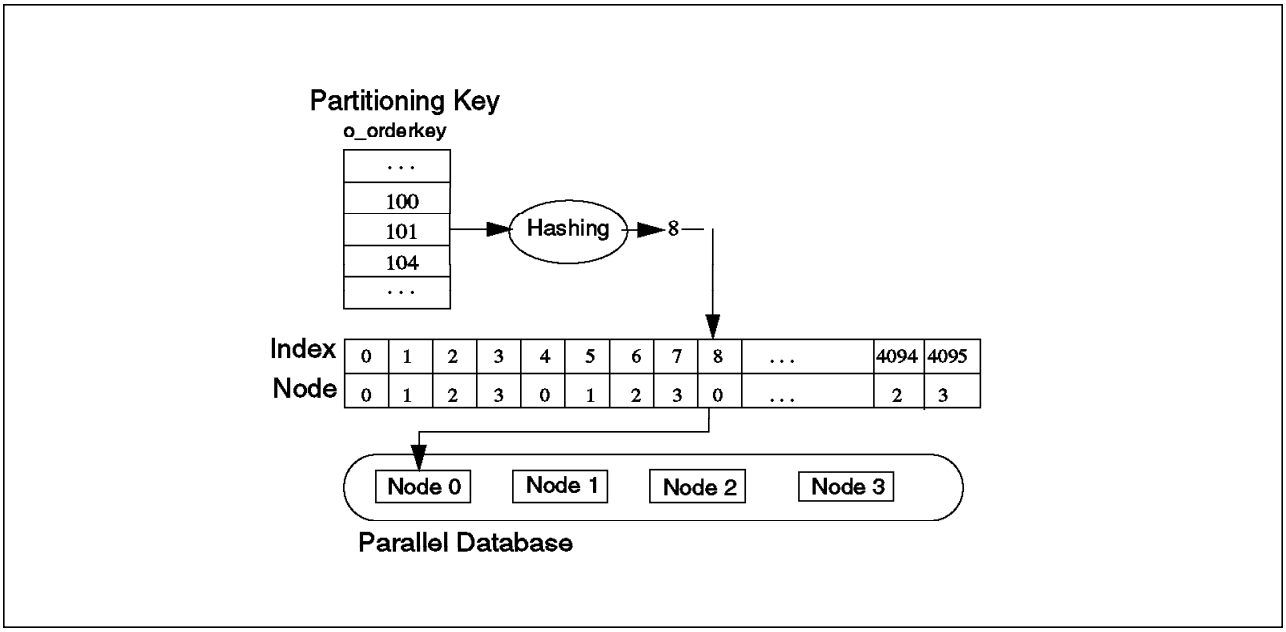


Figure 35. Partitioning Map

Figure 35 shows how a row with partitioning key (o_orderkey) value 101 is mapped to partition number 8.

3.7.6.1 Default Partition Map

The default partitioning map is created when a nodegroup is created. The map is stored in the SYSIBM.SYSPARTITIONMAPS catalog table.

The default partitioning map contains the node numbers of the nodegroup assigned in a round-robin fashion. Figure 36 illustrates the map for a nodegroup which contains four nodes numbered 0, 1, 2, and 3.

Index	0	1	2	3	4	5	6	7	8	...	4094	4095
Node	0	1	2	3	0	1	2	3	0	...	2	3

Figure 36. Default Partitioning Map

DB2 Parallel Edition provides a utility, db2gpmap, to extract the partitioning map of a table or nodegroup from the SYSIBM.SYSPARTITIONMAPS catalog table.

If the runtime module, db2gpmap, does not exist in the $HOME/sqllib/bin directory, you need to compile db2gpmap.sqc by:

- Copying the source materials from the $HOME/sqllib/samples/splitter directory to a work directory.
- Editing the db2split.make file to specify the database against which you want to preprocess the db2gpmap.sqc file.
- Executing the db2split.make file by issuing the following:

```
make -f db2split.make db2gpmap
```

You can issue the following command to show the syntax of db2gpmap:

```
db2gpmap -h
```

The following is an example of executing db2gpmap to get the partitioning map of the nodegroup, groupa, on the database dss, and to place the output map in the file groupa.pmap.

```
db2gpmap -d dss -m groupa.pmap -g groupa
```

3.7.6.2 Customized Partitioning Map

You may want to generate a customized map before loading data when populating a table in a new nodegroup. DB2 Parallel Edition provides a sample utility called db2split to analyze, and to partition the data file.

The db2split utility with the parameter (RunType=analyze) specified in the configuration file will generate a customized partitioning map based on the input nodes specified in the configuration file, and the input data file. It is recommended that you specify the input data file for the largest table in that nodegroup. The customized partitioning map is optimized to have an even data distribution over the nodes specified in the configuration file. After the customized partitioning map is generated, you may modify it using any editor. Be extremely cautious when you modify the customized partitioning map as it

may cause your data to be skewed. Skewing of data can adversely affect your system performance. When the customized partitioning map is ready, you will use the `redistribute nodegroup` command to catalog the map to the database before it can be used by the parallel database system. For more information on db2split and redistribute nodegroup, see Chapter 5, “Parallel Utilities” on page 141.

The following illustrates the steps used to generate and catalog a customized partitioning map:

- Modify the db2split configuration file for the input file, nodes, output mapfile, runtype, column delimiter, string delimiter, and partitioning key parameters.
- Run db2split in `RunType=analyze` mode to generate the customized partitioning map.
- Issue the `redistribute nodegroup` command to store the customized partitioning map in the SYSIBM.SYSPARTITIONMAPS catalog table.

3.7.6.3 Indirect Hashing

Figure 35 on page 67 shows the method that DB2 PE uses to determine the partition number. This is called the indirect hashing function. The hashing function ensures that the index value that is stored in the partitioning map will always evaluate to numbers between 0 and 4095. Further, it ensures that the data will be evenly spread among all nodes if the partitioning key is correctly defined. The hashing function takes the partitioning key as an input and applies the hashing algorithm to produce an index into the partitioning map that points to the correct node.

3.7.7 Row Partitioning

The steps for row partitioning consist of splitting and inserting the data across the target nodes.. DB2 Parallel Edition provides tools to partition the data and then load the data to the system via a fast-load utility or import. Data can also be inserted into the database via an application that uses buffered inserts.

Assuming the nodegroup and table are already defined to the database, the following illustrates the steps needed to partition and insert the data in the database:

- Get the partitioning map of the nodegroup or table by using the `db2gpmap` utility.
- Modify the db2split configuration file for the following parameters:
  - `Infile` for the input data file
  - `RecLen` for the record length of the input data file
  - `MapFili` for the input partitioning map from `db2gpmap`
  - `RunType=partition` for partition mode
  - `OutFile` for the prefix of the output file
  - `Partition` for partitioning key column
- Run `db2split` with the db2split configuration file to split or partition the data. For more information on `db2split`, see Chapter 5, “Parallel Utilities” on page 141.
- Insert the data to the database using the DB2 Parallel Edition load or import utility or a user application with buffered inserts. For more information on

the DB2 Parallel Edition utilities, see Chapter 5, “Parallel Utilities” on page 141.

Chapter 4. Parallel Processing

This chapter discusses the extensions of the major components in DB2 Parallel Edition to support parallel processing for all types of SQL operations. The major components discussed in this chapter are:

- SQL statements
- Not Initially Logged Tables
- Buffered inserts
- SQL operations
- CASE expressions
- OUTER JOIN operations
- DIGITS scalar function
- SQL optimization
- Explain tools
- Table queues
- Join strategies
- Locking

4.1 SQL Statements and Database Commands

There are two types of SQL statements:

- Data definition language (DDL)
- Data manipulation language (DML)

In addition, there are commands specific to DB2 that allow you to create or manage a database. For the complete syntax of all SQL statements covered in this chapter, refer to *DB2 Parallel Edition for AIX Administration Guide and Reference*.

4.1.1 Create/Drop Database

The `create database` command is used to create databases within DB2 Parallel Edition. A database can be created or dropped from any node that is configured in your parallel database system. The node on which the `create database` command is issued becomes the catalog node of the database.

Before you create your database, the database manager must be started on all the nodes of your parallel database system. After database creation, the system and local database directory can be listed using the `list database directory` command. For more information about the process of database creation and the database structure, refer to 3.5, “Database Creation” on page 49.

The following commands show how to create a database, dss, in the /database file system and display information about the database that is found in the system database directory and local database directory.

- Create a database called dss on /database directory.

```
create database dss on /database
```

- List the system database directory.

  ```
  list database directory
  ```

  Output:

  ```
          System Database Directory

     Number of entries in the directory = 1

   Database 1 entry:

    Database alias                 = DSS
    Database name                  = DSS
    Local database directory       = /database
    Database directory             =
    Node name                      =
    Database release level         = 7.00
    Comment                        =
    Directory entry type           = Indirect
    Authentication                 = SERVER
    Catalog node number            = 0
  ```

  The output from the system database directory indicates that node 0 is the catalog node. The catalog node is the node on which the create database command was issued.

- List the local database directory.

  ```
  list database directory on /database
  ```

  Output:

  ```
          Local Database Directory on /database

      Number of entries in the directory = 1

   Database 1 entry:

    Database alias                 = DSS
    Database name                  = DSS
    Local database directory       =
    Database directory             = SQL00001
    Node name                      =
    Database release level         = 7.00
    Comment                        =
    Directory entry type           = Home
    Authentication                 =
    Catalog node number            = 0
    Node number                    = 0
  ```

  The node number from the local database directory output indicates that the `list local database directory` command was issued from node 0.

Before you drop a database, the database manager must be started on all the nodes configured in the parallel database system and the database must exist on the system.

To drop an existing database, such as dss:

```
drop database dss
```

4.1.2 Data Definition Language (DDL)

Data Definition Language (DDL) is used to define and manipulate the structure of the database. The major DDL in Parallel Edition discussed in this section is:

- Create/drop nodegroup
- Create/drop and alter table
- Create/drop index

4.1.2.1 Create/Drop Nodegroup

The concept of a nodegroup is introduced in DB2 Parallel Edition to support table partitions. A nodegroup can be created and dropped from any configured node in the parallel database system. The maximum number of nodegroups currently supported in a database is 32,768. For more information about nodegroups, refer to 3.7.1, “Nodegroups” on page 60.

Before creating a nodegroup, the database manager must be started on all the nodes of your parallel database system and you must be connected to the database where the nodegroup will be created.

Consider a four node parallel database system with the following hostnames: HOST0, HOST1, HOST2, HOST3. Their corresponding node numbers are 0, 1, 2, and 3, respectively. The following examples will create nodegroups on the pdb system:

- Create a nodegroup ALLNODE on all four nodes.

```
db2 create nodegroup ALLNODE on all nodes
```

- Create a nodegroup NG3 on nodes 0, 1, and 3.

```
db2 create nodegroup NG3 on nodes (0 to 1, 3)
```

- Create a nodegroup NG1 on node 2.

```
db2 create nodegroup NG1 on node (2)
```

Before you drop a nodegroup, the database manager must be started on all the nodes of your parallel database system and you must connect to the database where you want the nodegroup to be dropped. The nodegroup must exist in the database.

For example, to drop an existing nodegroup, NG1:

```
db2 drop nodegroup NG1
```

Note: Tables in the nodegroup will be dropped without warning.

4.1.2.2 Create, Drop, or Alter Table

Tables can be created, dropped, and altered from any configured node that is in your parallel database system. The `drop table` SQL statement is the same as in DB2/6000.

The create table DDL is extended to support the concepts of partitioning keys and nodegroups. A primary key must be a superset of the partitioning key, if the table has a partitioning key. Before issuing the `create table` command, the nodegroup must already exist. The nodegroup can be either system-defined (IBMDEFAULTGROUP) or user-defined. The maximum size of a table at a node is 64 GB, or the available disk space, whichever is smaller.

Note: DB2 Parallel Edition currently does not support Referential Integrity (RI).

The following examples will create tables:

- Create a table called TAB in nodegroup NG4 with DATA_INT as the partitioning key.

```
create table TAB (DATA_CHAR char(20) not null,
                  DATA_INT integer not null,
                  DATA_DATE date not null)
                  in NG4
                  partitioning key(DATA_INT) using hashing
```

- Create a table called TAB_1 in the default nodegroup IBMDEFAULTGROUP and using the default partitioning key. In this example, the partitioning key will be the first non-long column, which is COL1.

```
create table TAB_1 (COL1 char(5) not null,
                    COL2 integer not null,
                    COL3 date)
```

The alter table SQL statement is extended to support partitioning keys. In DB2 Parallel Edition, you can use the alter table SQL statement to add or drop a partitioning key if the table resides in a single-node nodegroup. When adding a partitioning key, also ensure that:

- A partitioning key does not exist in the table
- All the rules for adding partitioning keys are followed

Note: Foreign keys are not supported. You will receive an error code of SQLCODE -270 if attempting to specify a foreign key.

The following are examples using the alter table SQL statement:

- Alter the table TAB_1 to add a partitioning key using column COL1. The table TAB_1 was created in a single-node nodegroup.

```
alter table TAB_1 add partitioning key(COL1)
```

- Alter the table BRANCH to drop its partitioning key. The table BRANCH was created in a multinode nodegroup NG.

```
alter table BRANCH drop partitioning key
```

  The following SQL error occurs:

```
SQL0264N  Partitioning key cannot be added or dropped because table
resides in the multinode nodegroup "NG". SQLSTATE=55037
```

4.1.2.3 Create/Drop Index

The commands to create and drop indices are the same as DB2/6000 with the following restriction:

- Definition of primary key or unique index

  A unique index or primary key must be a superset of the partitioning key, if any.

The indices are partitioned based on the partitioning key of the table.

4.1.3 Data Manipulation Language (DML)

Data Manipulation Language (DML) is used to query, populate, and modify the data in the database. The major DML statements are:

- Select
- Insert
- Update
- Delete

The DML used in DB2 Parallel Edition is the same as in DB2/6000 with the following restriction:

- Update of partitioning columns

  If a table resides in a multinode nodegroup, none of the partitioning columns can be specified for update by the UPDATE statement.

  To update the partitioning key value (or values), delete the old row and insert a new one.

A new option, called buffered insert, is introduced in DB2 Parallel Edition to improve the speed of the data insertion process. The buffered insert option will be discussed in 4.3, “Buffered Inserts” on page 78.

4.2 Not Initially Logged Tables

When large operations such as Insert, Delete, Update, and Create Index are done on tables, database logging might pose a problem because of:

- Log space requirements and limitations
- The time taken by logging

Because some tables are created and populated from master tables, the recoverability of these tables is not crucial. In the event of any errors, the tables can be recreated from the master tables.

In DB2 Parallel Edition V1.2, a clause ″NOT LOGGED INITIALLY″ is added to the `create table` SQL statement. For the table created with this clause, no logging is done for any changes made on the table (including Insert, Delete, Update, or Create Index operations) in the same unit of work where the table was created. This not only reduces the amount of database logging, but also may improve performance for your application. For the syntax of `create table`, see *DB2 Parallel Edition Administration Guide and Reference*.

Notes:

1. Changes to catalog tables made for creating a table with ″NOT LOGGED INITIALLY″ are still logged.
2. Any operations on the table in other units of work will be logged as usual.
3. A NOT LOGGED INITIALLY table is locked with a Superexclusive (Z) lock for the duration of the unit of work that creates it. This prevents concurrent access by an Uncommitted Read application.
4. The `commit` statement for a unit of work that creates a NOT LOGGED INITIALLY table waits until all of the buffer pool pages for the table have

been written to disk. For small tables, these synchronous writes may take longer than logging operations would have taken.

4.2.1 Considerations for Using Not Logged Initially

Because changes to the table are not logged, you should consider the following when creating a table with the ″NOT LOGGED INITIALLY″ clause:

- Since all the tables used in a static SQL application have to exist before an application can be bound, the NOT INITIALLY LOGGED feature can only be used with dynamic SQL.
- When a table is created with ″NOT LOGGED INITIALLY,″ a special record is written to the database log. When this log record is encountered during forward recovery, all the log records for this table will be ignored until a log record to drop this table is found. If such a record is not found, the table is marked as unavailable. After the database is recovered, any attempt to access the table will receive the following message:

  ```
  SQL1477N Table with id<id> cannot be accessed
  ```

  To solve the problem, user has to drop the table.
- Forward recovery cannot recover the table created with ″NOT LOGGED INITIALLY″ if the most recent database backup was taken before the creation of such table.
- If a statement that changes the table which was created with ″NOT LOGGED INITIALLY″ encounters an error, user may receive the following message and the unit of work will be rolled back.

  ```
  SQL1476N The current transaction was rolled back because of
           error<error code> on the table with id<id>
  ```

The following illustrates the effect of errors on the statements in the scenario where two logical unit of works execute one after another.

```
    ******** First unit of work begin **********
(1)  Create table t1
(2)  Create table t2 with not logged initially
(3)  Insert into t2 select from s2
(4)  Insert into t1 select from s1
(5)  Create table t3 with not logged initially
(6)  Create index on t2
(7)  Commit
    ******** First unit of work end **********
    ******** Second unit of work begin **********
(8)  Create table t4 with not logged initially
(9)  Insert into t4 select from s4
(10) Insert into t3 select from t4
(11) Commit
    ******** Second unit of work end **********
```

- When an error occurs on the statement in line **3**, a UNIT OF WORK ROLLBACK will be done. The following works will be affected:
  - The creation of table t1 in line **1**.
  - The creation of table t2 in line **2**.
  - The insert operation into t2 in line **3**.

- When an error occurs on the statement in line **4**, a STATEMENT ROLLBACK will be done. Only the insert operation into t1 in line **4** will be rolled back.
- When an error occurs on the statement in line **6**, a UNIT OF WORK ROLLBACK will be done. The following work will be affected:
  - The creation of table t1 in line **1**.
  - The creation of table t2 in line **2**.
  - The insert operation into t2 in line **3**.
  - The insert operation into t1 in line **4**.
  - The creation of table t3 in line **5**.
  - The index creation for table t2 in line **6**.
- When an error occurs on the statement in line **9**, a UNIT OF WORK ROLLBACK will be done. Since this statement is executed in the second unit of work, only the creation of table t4 in line **8** will be rolled back.
- When an error occurs on the statement in line **10**, a STATEMENT ROLLBACK will be done. Only the insert operation into t3 in line **10** will be rolled back.

4.2.2 Example

The following example shows that an error occurred within a logical unit of work during index creation on a table created with the ″NOT LOGGED INITIALLY″ clause.

1. Place the following commands into a file called nologi.test.

```
CONNECT TO dss;
CREATE TABLE t2 (col1 INTEGER) NOT LOGGED INITIALLY;
INSERT INTO t2 SELECT * FROM t1;
CREATE UNIQUE INDEX i2 ON t2 (col1);
COMMIT;
```

2. Execute the previous commands using DB2 command line with autocommit off and send the output message to the file nologi.test.out.

```
db2 +c -tvf nologi.test > nologi.test.out
```

3. The message SQL1476 indicates the UNIT OF WORK ROLLBACK has occurred during index creation on table with FID 24. The error ″-603″ indicates a unique index cannot be created due to duplicate rows. The following shows the content of nologi.test.out file.

```
connect to dss

   Database Connection Information

 Database product       = DB2/6000 PE 1.2.0
 SQL authorization ID   = DB2PE
 Local database alias   = DSS


create table t2 (col1 integer) not logged initially
DB20000I  The SQL command completed successfully.

insert into t2 select * from t1
DB20000I  The SQL command completed successfully.

create unique index i2 on t2 (col1)
DB21034E  The command was processed as an SQL statement and returned:
SQL1476N  The current transaction was rolled back because of error "-603" on
the table with id "24".  SQLSTATE=40506

commit
DB20000I  The SQL command completed successfully.
```

4.3 Buffered Inserts

In DB2 Parallel Edition, rows can be inserted using a buffered or non-buffered option. The new buffered insert option utilizes table queue services to achieve parallelism for performance. For more information about table queues, refer to 4.7, “Table Queues” on page 100.

For non-buffered insertion, DB2 Parallel Edition inserts one row at a time to the database. This is the default option for row insertion in DB2 Parallel Edition. See Figure 37 for an illustration of this mechanism.

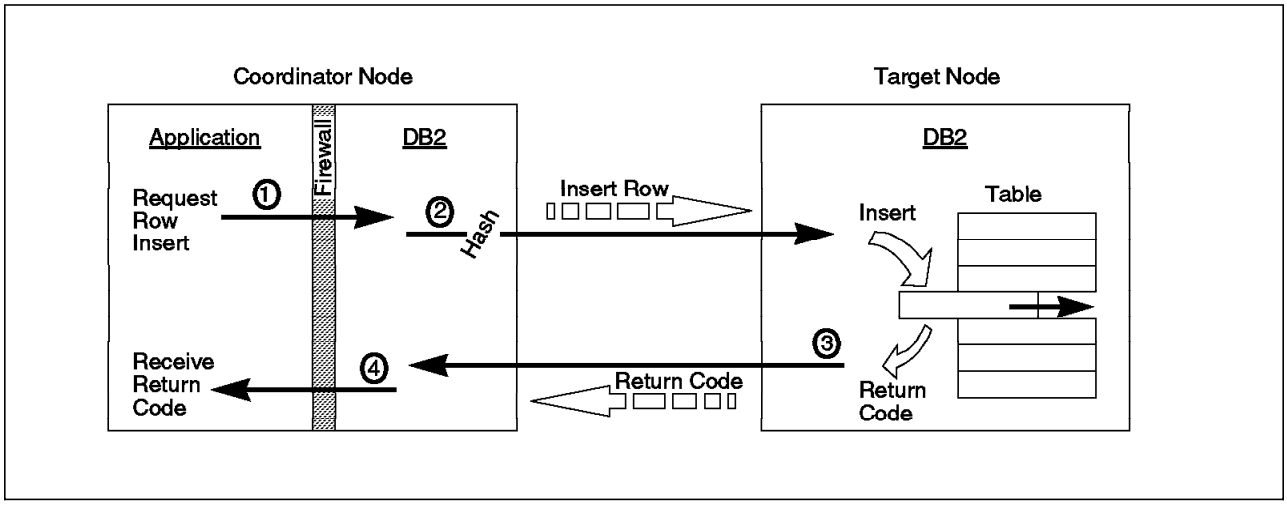


Figure 37. Non-Buffered Row Insertion

The steps as defined in Figure 37 are as follows:

1. (Application) Sends the row across the firewall to the database manager on the same node.

2. (Coordinator node) Database manager uses the partitioning key of that row and applies the hashing algorithm to determine what node is the target node. The database manager then sends the row to the target node.
3. (Target node) Receives and inserts the row, then sends response to the coordinator.
4. (Coordinator node) Receives response from the target node, and presents the response to the application.

   Note: The insertion is committed only when the application issues a commit.

For buffered insertion, DB2 Parallel Edition packs the rows into a buffer before insertion. As a result, the following performance advantages are achieved:

- For each buffer received by the target node, there is only one message sent from the target node to the coordinator node.
- If the size of the row is small, a buffer can contain a large number of rows.
- The target nodes performing insertions are processed in parallel with the coordinator node receiving new rows.

Assuming the application is executing locally at one of the nodes, the following illustrates how DB2 Parallel Edition handles the buffered insertion:

1. (Application) Sends the row across the firewall to the database manager on the same node.
2. (Coordinator node) Database manager opens one 4 KB table queue buffer for each node of the target table. The database manager uses the row′s partitioning key as an input for the hashing algorithm to determine on which node the row should be placed. It places the row into the buffer of the appropriate node. It then returns control to the application.

   The table queue buffers are closed and the rows in the buffers are sent to the nodes when one of the following condition occurs:

   - The buffer becomes full
   - The application issues an implicit or explicit commit
   - An updating statement is invoked while a buffered INSERT statement is open. The following statements are updating statements:
     - INSERT
     - UPDATE
     - DELETE
     - DDL
     - GRANT
     - REVOKE
     - REORG
     - RUNSTATS
     - REDISTRIBUTE NODEGROUP
     - SELECT INTO
     - BEGIN COMPOUND SQL
     - END COMPOUND SQL
     - EXECUTE IMMEDIATE

- A PREPARE statement is issued

3. (Target node) Receives the table queue buffer, takes the rows from the buffer, and inserts them into the target table. Sends a message for the received buffer to the coordinator.

   (Coordinator node) Continues to receive new rows from the application.

Note: The coordinator node waits until the table queue buffers are received and the rows are inserted successfully by every node before completing a commit or before executing an updating statement.

4.3.1 Enabling the Buffered Insert Option

By default, INSERT with VALUES inserts one row at a time. To enable buffered insertion, you must PREP or BIND your application with INSERT BUF option. The option can be set using:

- Command Line Processor
- Application Programming Interface

4.3.2 Considerations for Using Buffered Insert

In order to use buffered inserts for parallelism and performance advantages, consider designing your application so that the same INSERT with VALUES statement can be iterated repeatedly before any of the following commands are issued:

- A COMMIT
- An EXEC IMMED
- A different EXEC

You must ensure that you have enough space in the log files for the insertion. Otherwise, issue periodic commits to prevent the log files from filling up from the insertion process.

4.3.3 Restrictions

The following restrictions apply:

- Buffered inserts can be used only in user applications, not from the Command Line Processor.
- The buffered insert option is ignored when one of the following is encountered:
  - INSERT with VALUES including any long fields in the explicit or implicit column list.
  - INSERT with full-select.

4.4 SQL Operations

This section describes other parallel operations running under DB2 Parallel Edition.

4.4.1 Set Operations

The set operations UNION, EXCEPT, and INTERSECT will be performed locally in parallel if possible, or at the coordinator node if not. The following are requirements for the set operations to work in parallel:

Operation	Conditions
UNION ALL	Tables on both sides of the UNION ALL command must be in the same node group.
UNION EXCEPT EXCEPT ALL INTERSECT INTERSECT ALL	The following are all requirements for parallelism to work: • Tables on both sides of the command must be collocated. • All partitioning columns must be directly selected on both sides of the operation. • All partitioning key columns of the left hand side table must appear in the same position as the corresponding partitioning key columns on the right hand side of the operation.

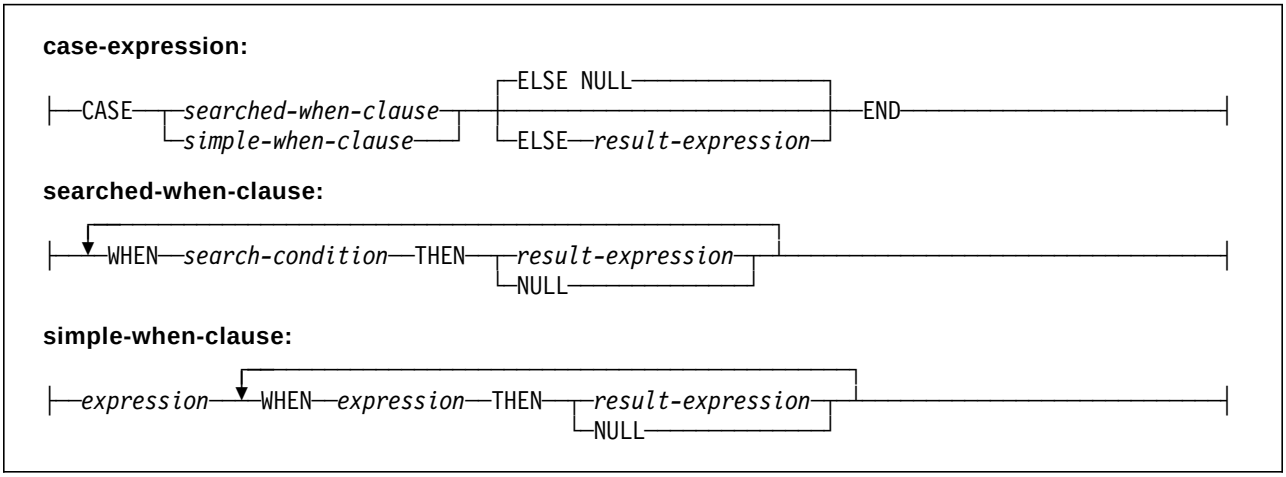
4.4.2 Group By Operations

The GROUP BY operation is performed locally if all partitioning columns participate in the GROUP BY list. If not, then the GROUP BY is done partially at the local node and merged at the coordinator node. The DISTINCT operator may disallow even the partial local grouping.

4.4.3 CASE Expressions

CASE expressions are supported in DB2 Parallel Edition V1.2. They allow an expression to be selected based on the evaluation of one or more conditions.

The syntax is as follows:



Note: All keywords of both simple- and searched-CASE expressions (i.e., *CASE*, *WHEN*, *THEN*, *ELSE*, and *END*) are reserved.

The value of the CASE expression is determined by:

- The value of the result-expression following the first (leftmost) case that evaluates to true.

- If no search condition evaluates to true and the ELSE clause is present, the result is the value of the ELSE clause.
- If no search condition evaluates to true and the ELSE clause is not present, the result is NULL.

4.4.3.1 Restrictions

- The search-condition in a searched-when-clause cannot be a predicate that includes a subquery. If violated, the following SQL error with SQLCODE -582 and SQLSTATE 42625 will occur:

  ```
  SQL0582N: A CASE expression cannot include a predicate which
            contains a subquery.
  ```

- All result-expressions must have (SQL) compatible data types. Otherwise, the following SQL error with SQLCODE -581 and SQLSTATE 42804 will occur:

  ```
  SQL0581N: The data types of the result-expressions of a CASE
            expression are not compatible.
  ```

- At least one non-NULL result-expression must exist in the CASE expression. Otherwise, the following SQL error with SQLCODE -580 and SQLSTATE 42625 will occur:

  ```
  SQL0580N: The result-expression of a CASE expression cannot all be
            NULL.
  ```

Example: The following example shows a CASE expression can be used to translate cryptic codes into meaningful labels. Select the employee number, last name, and a descriptive work-department name based on the first letter of the work-department (WORKDEPT).

```
SELECT emp, lastname,
       CASE SUBSTR(workdept, 1, 1)
            WHEN 'A' THEN 'Accounting'
            WHEN 'H' THEN 'Human Resources'
            WHEN 'D' THEN 'Development'
       END
FROM employee
```

4.4.4 Outer Join

DB2 Parallel Edition V1.1 only supported join operations from two tables where the matching values in the join columns have the same value. If a row of a table is unmatched, that row is omitted from the result table. This join operation is called ***INNER JOIN***. To retain the unmatched row, the user would need to rewrite the query.

To understand the shortcomings of **INNER JOIN**, consider the following tables, 'department,' and 'employee':

```
    Department:                 Employee:

DEPT   DEPTNAME            EMP  FNAME           WDEPT
------ ---------           ---- --------------- -----
A01    Acc                 10   Frank           B01
B01    Plan                20   Randy           C01
C01    Dev                 30   Gus             C01
D01    Info                40   Jean-Christophe C01
```

```
                                50   Michael         C01
```

The join (INNER JOIN) of these two tables:

```
SELECT dept, deptname, fname
FROM  department, employee
WHERE dept = wdept

       or

SELECT dept, deptname, fname
FROM  department INNER JOIN employee
ON dept = wdept
```

The join (INNER JOIN) of tables 'department' and 'employee' produces the following result table:

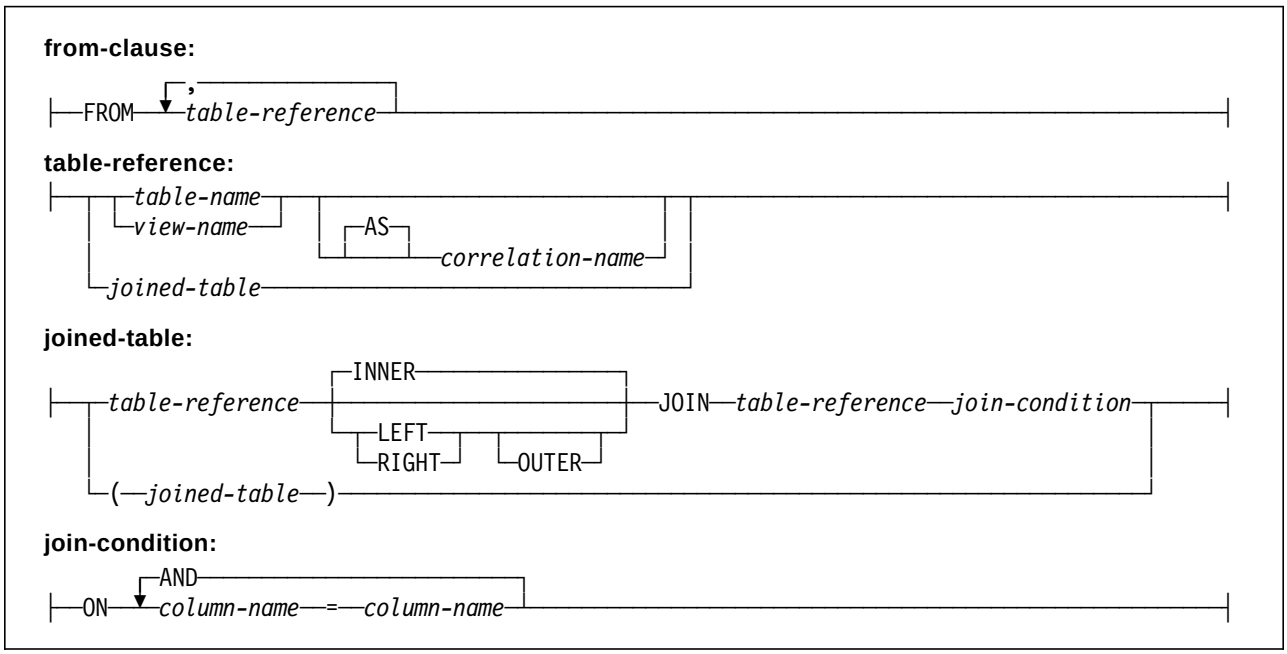
```
DEPT    DEPTNAME FNAME
------  -------  ----------------
B01     Plan     Frank
C01     Dev      Randy
C01     Dev      Gus
C01     Dev      Jean-Christophe
C01     Dev      Michael
```

Note: There are no rows corresponding to the departments with DEPT values of A01, or D01 in the result table.

OUTER JOINs are a set of join operators in which the join condition specifies a paring that is not also a restriction. There are three types of outer join:

- ***LEFT OUTER JOIN*** which includes the matched rows and preserves the unmatched rows of the left operand table.
- ***RIGHT OUTER JOIN*** which includes the matched rows and preserves the unmatched rows of the right operand table.
- ***FULL OUTER JOIN*** which includes the matched rows and preserves the unmatched rows of both tables.

DB2 Parallel Edition in V1.2 only supports two types of outer join: ***LEFT OUTER JOIN*** and ***RIGHT OUTER JOIN***. The existing FROM clause of the subselect is enhanced as depicted below:



4.4.4.1 Considerations for Using OUTER JOIN

The following should be considered when using OUTER JOIN:

- There may be more rows returned as a result of the OUTER JOIN.
- If a join-operator is not specified, ***INNER JOIN*** will be used implicitly.
- Both ***LEFT OUTER JOIN*** and ***RIGHT OUTER JOIN*** are not associative. This means that the order in which ***LEFT OUTER JOIN*** or ***RIGHT OUTER JOIN*** operations are performed is important and it may affect the result of the join operation.
- When OUTER JOIN is used, the outer table is determined by:
  - The left operand of ***LEFT OUTER JOIN***
  - The right operand of ***RIGHT OUTER JOIN***

  The SQL optimizer does not switch the inner and outer tables when choosing the best execution plan. This must be taken into consideration when choosing the partitioning key of the tables involved in the OUTER JOIN operations.

4.4.4.2 Restrictions

The following restrictions apply:

- ***FULL OUTER JOIN*** is NOT supported in DB2 Parallel Edition V1.2.
- Only column names of the joined tables can be referenced as the predicates in the join condition. Each basic predicate must be in the form `column1 = column2` where column1 refers to a column in one of the operand joined tables and column2 refers to a column in the other operand joined table.
- Only the '=' operator is allowed in the predicates of the join-condition and only the AND operator can be used to combine them.

Example: The following example joins tables 'department' and 'employee' using ***LEFT OUTER JOIN*** to list the department and employee that are in the same working department along with any unmatched department.

```
     Department:                    Employee:

  DEPT    DEPTNAME              EMP  FNAME           WDEPT
  ------  ---------             ---- --------------- -----
  A01     Acc                   10   Frank           B01
  B01     Plan                  20   Randy           C01
  C01     Dev                   30   Gus             C01
  D01     Info                  40   Jean-Christophe C01
                                50   Michael         C01
```

The join (LEFT OUTER JOIN) of these two tables:

```
SELECT dept, deptname, fname
FROM  department LEFT OUTER JOIN employee
ON dept = wdept
```

The join (LEFT OUTER JOIN) of tables 'department' and 'employee' produces the following result table:

```
DEPT    DEPTNAME FNAME
------  -------  ----------------
A01     Acc      -
C01     Dev      Randy
C01     Dev      Gus
C01     Dev      Jean-Christophe
C01     Dev      Michael
B01     Plan     Frank
D01     Info     -
```

Note: The rows for the table 'department' with DEPT values of (A01, D01) in the result table indicates there is no employee in those departments because the value for FNAME is NULL.

The SQL optimizer will convert an outer join to an inner join if the predicates in the where clause disallow NULL values in the results:

```
SELECT dept, deptname, fname
FROM  department LEFT OUTER JOIN employee
ON dept = wdept
Where fname between 'Adam' and 'Zelda'
```

is treated the same as:

```
SELECT dept, deptname, fname
FROM  department INNER JOIN employee
ON dept = wdept
Where fname between 'Adam' and 'Zelda'
```

and produces the following output:

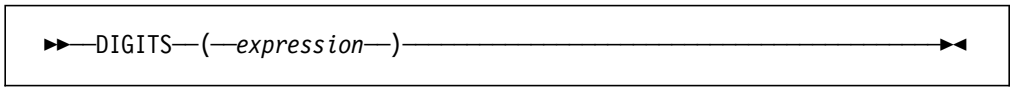
```
DEPT    DEPTNAME FNAME
------  -------  ---------------
C01     Dev      Randy
C01     Dev      Gus
C01     Dev      Jean-Christophe
C01     Dev      Michael
B01     Plan     Frank
```

4.4.5 DIGITS Scalar Function

The DIGITS function is supported in DB2 Parallel Edition V1.2. It transforms the numeric value of an argument into a character-string representation of that number without regard to its scale or sign. The results is a character-string which only consists of digits and does not include a sign or a decimal character. The length of the string is:

- 5 if the argument is a small integer
- 10 if the argument is a large integer
- p if the argument is a decimal number with a precision of p

The syntax is as follows:



Note: The argument of the DIGITS function must be an expression that returns a value of type SMALLINT, INTEGER, or DECIMAL.

Example

- The following is the create table definition of table tab1:

  ```
  CREATE TABLE tab1 (colint INTEGER, coldec DEC(6,2))
  ```

- Select the rows from table tab1 and display the result:

  ```
  SELECT * FROM tab1

  RESULT:

    COLINT      COLDEC
    ----------- --------
              -     2.00
              1    10.50
              2   -10.00
  ```

- Select the rows from table tab1 using DIGITS function:

```
SELECT DIGITS(colint), DIGITS(coldec), DIGITS(colint+1) FROM tab1

RESULT:

  1          2      3
  ---------- ------ ----------
  0000000001 001050 0000000002
  -          000200 -
  0000000002 001000 0000000003
```

Note: The above example shows:

- The result of the DIGITS function does not include a sign or a decimal character.
- If the value of the argument is null, the result is the null value.

4.4.6 SQL Functions

There are two other SQL functions and one special register specific to DB2 Parallel Edition:

- NODENUMBER
- PARTITION
- CURRENT NODE

The NODENUMBER function takes the column name as input to get information about the node number of the row. The column name used in the argument must be an actual column name of the table and must not refer to a view. The column name in the argument of this function must not be from the inner (null supplying) table of an outer join, otherwise SQLCODE -270 is issued. The result of the NODENUMBER function is an integer.

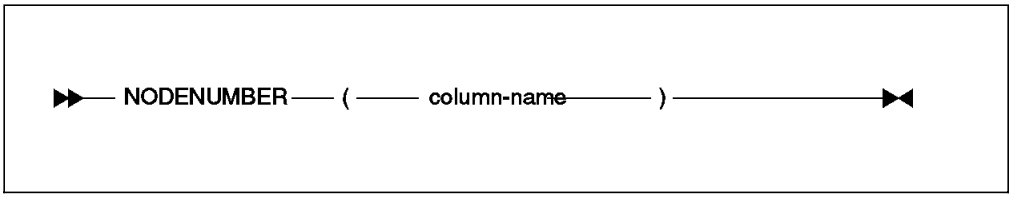


Figure 38. NODENUMBER Syntax

Example

```
SELECT L_ORDERKEY, NODENUMBER(L_ORDERKEY)
  FROM LINEITEM
```

The PARTITION function takes the column name as input to get the information about the partition number of the row. The column name used in the argument must be an actual column name of the table and must not refer to a view. The column name in the argument of this function must not be from the inner (null supplying) table of an outer join, otherwise SQLCODE -270 is issued. The result of the PARTITION function is an integer in the range of 0 to 4095.

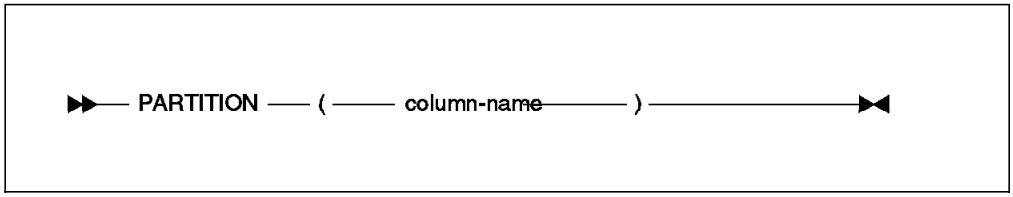


Figure 39. PARTITION Syntax

Example

```
SELECT L_ORDERKEY, PARTITION(L_ORDERKEY)
  FROM LINEITEM
```

The CURRENT NODE special register specifies an integer that identifies the coordinator node. This value can be used in expressions exactly as other special registers.

4.4.7 Column Functions

The column functions MAX, MIN, SUM, AVG, and COUNT are performed locally, or locally and globally. Completely local processing is done if all partitioning columns are also grouping columns. Otherwise partial results are formed locally, and then shipped to the consumer or coordinator node for collation.

The column functions SUM, AVG, and COUNT with DISTINCT are performed locally, globally, or locally and globally. Completely local processing is done if all partitioning columns are also grouping columns. Completely global processing is done if the DISTINCT operator is applied to a column that is not the single-column partitioning key, and the grouping columns do not include all columns of the partitioning key. Otherwise partial results are formed locally, and then shipped to the consumer or coordinator node for collation.

The specification of DISTINCT has not effect on the result of the MIN or MAX column functions, and therefore, is not recommended. It is included for compatibility with other relational systems.

4.5 SQL Optimization

The SQL Optimizer in DB2 Parallel Edition is extended to generate parallel plans. DB2 Parallel Edition uses a cost-based query optimizer, which chooses the lowest-cost execution plan of a query. The information used by the optimizer to calculate the lowest-cost execution plans is:

- Table partitioning

  The optimizer uses the information about how the base tables are partitioned across nodes to determine the best execution strategies. The table partition is also important for the optimizer to determine the best join strategy. The join strategies include collocated, directed, and broadcast joins.

- Table statistics

  The optimizer uses the information in the catalog tables to calculate the lowest-cost access path.

- Index statistics

  The optimizer uses the information to determine if an index scan is more efficient than a table scan in terms of cost.

- Database configuration parameters

  The optimizer uses the database configuration parameters of the coordinator node to determine the cost of sorting and whether a page will remain in the buffer.

- SQL query

  The optimizer evaluates each query differently depending on the structure of the query.

4.6 Explain Tools

DB2 Parallel Edition provides two tools to analyze the access paths of the SQL statements:

- The db2expln command is used to explain static SQL statements. The following is the syntax:

  ```
  db2expln -d dbname -c creator -p package-name -s section-number
  ```

- The dynexpln command is used to explain dynamic SQL statements. The following is the syntax:

  ```
  dynexpln dbname "query"
  ```

  or

  ```
  dynexpln dbname "query" > query.expln.out
  ```

  The output of the dynamic explain can be sent to the standard output device or redirected to a file. If you have a long query, you should create a script to contain the query. For example, the following query is in a file called query.input

  ```
  select t1.c1, t.c2, sum(t2.c3)
   from t1, t2
   where t1.c1 = t2.c1 and t2.c3 < 100
   group by t1.c1, t1.c2
  ```

  You would execute the dynexpln in the following way to get the explain output.

  ```
  dynexpln dbname "cat query.input" > query.expln.out
  ```

The dynexpln shell script treats any SQL statement with SELECT as the first token on a line as a SELECT statement. This can cause unexpected failures during the "C" precompiler step on INSERT, DELETE, and UPDATE statements with subqueries, because the shell script declares a cursor:

```
prep dynexpln.sqc PACKAGE

LINE    MESSAGES FOR dynexpln.sqc
------  --------------------------------------------------------------------
        SQL0060W  The "C" precompiler is in progress.
    2   SQL0199N  The use of the reserved word "insert" following
                  "FOR" is not valid.  Expected tokens may include:
                  "ACQUIRE".
        SQL0092N  No package was created because of previous
                  errors.
        SQL0091W  Precompilation or binding was ended with "2"
                  errors and "0" warnings.
```

These SQL statements can be explained after joining the line containing the SELECT with the line above it.

4.6.1 Example of Explain Report

The following query performs a select to retrieve all order keys from the ORDERS table where the order date is after the first of July 1993.

```
SELECT O_ORDERKEY FROM ORDERS
WHERE O_ORDERDATE > DATE('1993-07-01')
ORDER BY O_ORDERKEY
```

The explain for this query is given in Figure 40.

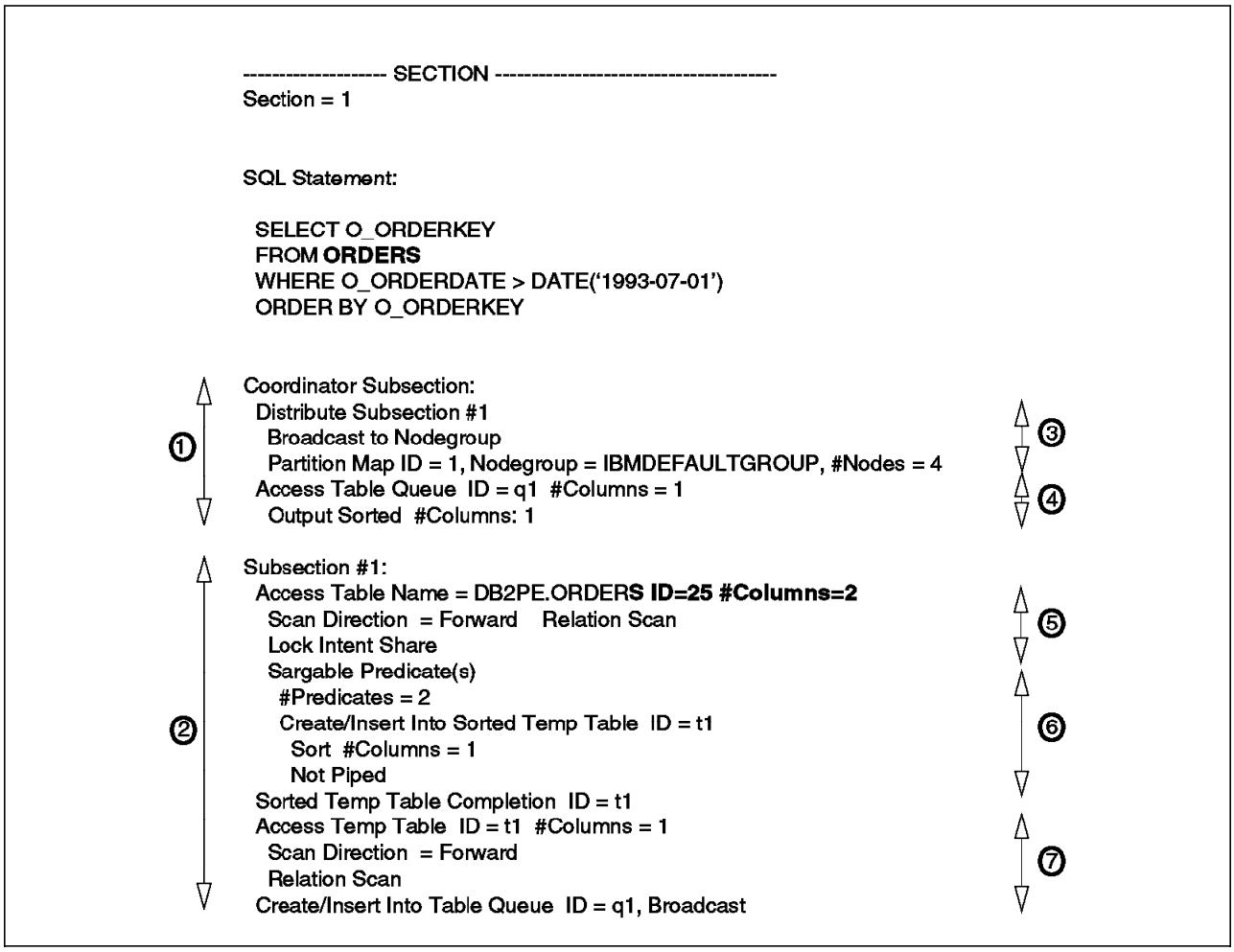


Figure 40. Explain Example

There are two subsections in the example. The part labelled 1 is the coordinator subsection, and always exists. The following is an explanation of the parts of this section:

- The first part of this subsection labeled 3 shows that subsection #1 will be executed on all nodes of the specified nodegroup. A broadcast routing method has been chosen by DB2 Parallel Edition which means that subsection #1 will go to all nodes in the nodegroup. The nodegroup is

specified as IBMDEFAULTGROUP, and the number of nodes in this nodegroup is 4.

- The second part, labeled 4, of this subsection specifies how the coordinator will process returned values. In this example the output is read from a table queue, q1. These values have been sorted on the first column. This column is O_ORDERKEY.

The second subsection is labeled 2, and will run on all nodes in the IBMDEFAULTGROUP nodegroup, as indicated in the coordinator subsection. The different parts of this subsection are described below.

- The first part, labeled 5, specifies that the DB2PE.ORDERS table will be accessed, and two of its columns retrieved. The table has an internal file identifier of 25. There will be a forward relational scan. The table will be locked in Intent Share mode.
- The second part, labeled 6, specifies that the predicates in the WHERE clause will be evaluated by the database manager as soon as the row is read by the relation scan. As part of the predicate processing, each row that meets the criteria of the predicates is added to a sorted temporary table, t1. The column to be sorted is considered a predicate, thus #predicates = 2.
- In the final part, labeled 7, the temporary table t1 is scanned in a forward direction. Rows will be added to a table queue q1 which are then sent to the coordinator node.

4.6.2 Description of Explain Report

This section describes the report generated by the explain tool. The report shows the parallel plan chosen by the DB2 Parallel Edition optimizer to execute the SQL statement.

When reading the explain report, you should start with the subsections which have the ″base″ tables. ″Base″ table means the table being referenced in the SQL statement, not the temporary table names, or table queue names.

4.6.2.1 Subsection Statements

In DB2 Parallel Edition, each parallel plan consists of at least two subsections:

- Coordinator subsection which does the process control. This subsection is executed on the coordinator node by the coordinating agent (db2agent).
- Subsections which are started by the coordinating agent and communicate with the coordinator subsection and other subsections via table queues.

Coordinator Subsection: This is the heading for subsection 0, which always exists. It gets control when an SQL request is issued from an application. This subsection is executed on the coordinator node. It distributes the request to other subsections, and returns results to the application.

Subsection #n: This text is the heading provided for each parallel subsection, which are uniquely numbered. These subsections are distributed and invoked by the coordinator subsection to perform various tasks in the execution of an SQL request. Each parallel subsection on each node is executed by a parallel agent process, db2agntp (unless locally bypassable). There may be more than one parallel agent at a given node working on behalf of an SQL request because multiple subsections may be required.

Distribute Subsection #n: This text shows if a subsection is to be executed at one or all nodes of a nodegroup. The sub-text of the following distribution information can accompany the subsection text:

- Information that indicates if the subsection is locally bypassable. This enhancement in DB2 Parallel Edition V1.2 has been implemented to improve performance for a subsection which is executed locally on the coordinator node. The improvement is to have the coordinating agent (db2agent) execute the query rather than sending the subsection to the PDB system controller and acquiring a parallel agent (db2agntp) to do the work.

  ```
  Locally bypassable
  ```

- Routing method:

  ```
  Broadcast to Nodegroup
     (to all nodes of a nodegroup)
  Directed to Single Node
     (to a single node)
  Directed by Hash #Columns = n
     (to one node of a nodegroup based on a value)
  Directed by Position
     (to the node that provided a cursor's current row)
  ```

- Partitioning information that applies to all routing methods except Directed by Position:

  ```
  Partitioning Map ID = n
  ```

- Nodegroup Name:

  ```
  Nodegroup = xxxxxx
  ```

- Number of nodes:

  ```
  #Nodes = n
  ```

4.6.2.2 Table Access Statements

This text description is the same as that found in DB2/6000. This statement tells the name and type of table being accessed. It has two formats that are used:

1. Regular tables:

   ```
   Access Table Name = qualifier.tbname  ID = xx  #Columns = yy
   ```

   where:

   - `qualifier.tbname` is the fully-qualified name of the table being accessed
   - `ID` is the corresponding FID column in the SYSIBM.SYSTABLES catalog table
   - `#Columns` indicates the number of columns being used from each row of the table

2. Temporary tables:

   ```
   Access Temp Table ID = xx  #Columns = yy
   ```

   where:

   - `ID` is the corresponding identifier assigned by db2expln
   - `#Columns` indicates the number of columns in each row of the table

Following the table access statement, additional statements will be provided to further describe the access. These statements will be indented under the table access statement. The possible statements are:

- Scan Direction

- Row Access Method
- Locking Mode
- Predicates

Scan Direction: The scan direction can either be forward or reverse. Note that an index scan can only read data in forward order. The format is:

```
Scan Direction = xxxxxx
```

Row Access Method: In DB2 Parallel Edition, scan operations can be executed in parallel across nodes. In the explain report, one of the following statements indicate how the qualifying rows in the table will be accessed:

- Relation Scan indicates that the table is being sequentially scanned to find the qualifying rows.
- Index Scan indicates that the qualifying rows are being identified and accessed through an index:

  ```
  Index Scan:  Name = qualifier.idxname  ID = xx  #Key Columns = yy
  ```

  where:

  - qualifier.idxname is the fully-qualified name of the index being scanned.
  - ID is the corresponding ID column in the SYSIBM.SYSINDEXES catalog table.
  - #Key Columns indicates the number of range-delimiting predicates, that is, the number of columns in the index key (from left to right) being used to delimit the index scan range. If #Key Columns = 0, a full scan of the index is being performed.

    If all the needed columns can be obtained from the index key,

    ```
    Index-only Access
    ```

    will appear and no table data will be accessed.

    If there are predicates that can be passed to the Index Manager to help qualify the index entries, the following statement is used to show the number of such predicates:

    ```
    Sargable Index Predicate(s)
       #Predicates = n
    ```

- The Fetch Direct statement indicates that the qualifying rows are being accessed by using row IDs (RIDs) that were prepared earlier in the access plan.
- The Table-in-Memory statement indicates that the qualifying rows are being identified and accessed through a binary search of a sorted temporary table that is resident in memory.

Lock Mode: For each table access, the type of lock that will be acquired is shown with one of the following statements:

- Lock Exclusive
- Lock Intent Exclusive
- Lock Intent Exclusive Immediate
- Lock Intent Exclusive with U row locks
- Lock Intent None
- Lock Intent Share
- Lock Share
- Lock Share Intent Exclusive

- Lock Super Exclusive
- Lock U (Update) For more information about lock modes, see 4.9, “Database Locking” on page 126.

Predicates: There are three statements that provide information about the predicates used in an access plan:

1. The following statement indicates the number of predicates that will be evaluated once the data has been returned:

   ```
   Residual Predicate(s)
      #Predicates = n
   ```

2. The following statement indicates the number of predicates that will be evaluated while the data is being accessed:

   ```
   Sargable Predicate(s)
      #Predicates = n
   ```

3. Since predicates concerning the inner table in the outer join may be correctly applied only after the join is performed, the following statement may appear in the explain output:

   ```
   Outer Join Residual Predicate(s)
      #Predicates = n
   ```

The number of predicates shown in the above statements may not reflect the number of predicates provided in the SQL statement because predicates can:

- Be applied more than once within the same query
- Be transformed and extended with the addition of implicit predicates during the query optimization process
- Be transformed and condensed into fewer predicates during the query optimization process
- Include push-down operations such as aggregation or sort

4.6.2.3 Temporary Tables

A temporary table is used by an access plan to store data during its execution in a transient or temporary work table. The table only exists while the access plan is being executed. Generally, temporary tables are used when subqueries need to be evaluated early in the access plan, or when intermediate results will not fit in the available memory.

If a temporary table needs to be created, then one of two possible statements may appear. These statements indicate that a temporary table is to be created and rows inserted into it. The ID is an identifier assigned by db2expln for convenience when referring to the temporary table. This ID is prefixed with the letter ’t’ to indicate that the table is a temporary table.

1. The following statement indicates an ordinary temporary table will be created: `Create/Insert Into Temp Table ID = tn`
2. The following statement indicates a sorted temporary table will be created: `Create/Insert Into Sorted Temp Table ID = tn`

A sorted temporary table is created through the use of the Sort Services. Sorts can result from such operations as:

- ORDER BY
- DISTINCT

- GROUP BY
- Merge Join
- '= ANY' subquery
- '< > ALL' subquery
- INTERSECT or EXCEPT
- UNION (without the ALL keyword)

A number of additional statements may follow the original creation statement for a sorted temporary table:

- The following statement indicates the number of key columns used in the sort:

  ```
  Sort #Columns = n
  ```

- The following statements indicate whether or not the results from the sort can be left in the sort heap:

  ```
  Piped
  ```

  and

  ```
  Not Piped
  ```

  A sort cannot be piped if the result from the sort will be used as the inner table of a join. Inner tables of joins need to allow for possible re-scanning, which would not be possible with piped sort output.

  At execution time a request for a piped sort may or may not be granted based on the currently allocated sortheap, the sheapthres database manager parameter, and the setting of the DB2_SORT_CUSHION_FOR_PIPE environment variable. See the $HOME/sqllib/Readme/$LANG/README file for details.

  The Database System Monitor can be used to determine whether or not piped sort requests have been accepted at execution time:

  ```
  db2 get snapshot for database manager | grep -i sort
  ```

  This output shows that all of the piped sort requests were accepted by the database manager:

  ```
  Sort heap allocated                         = 0
  Post threshold sorts                        = Not Collected
  Piped sorts requested                       = 3
  Piped sorts accepted                        = 3
  ```

- The following statement indicates that duplicate values will be removed during the sort:

  ```
  Duplicate Elimination
  ```

- The following statement indicates that one or more aggregation predicates will be partially applied during the sort:

  ```
  Aggregation in Sort
  ```

  It is possible that more than one row for the same aggregation key (for example, grouping columns) is output, and therefore, there will always be a subsequent aggregation completion. This method is not used if there are any column functions which contain the DISTINCT keyword.

- Once a temporary table is created and rows inserted into it, it is possible that more rows will be inserted into it. If this is the case, one of the following statements will appear:

  ```
  Insert Into Temp Table  ID = tn
  ```

  or

  ```
  Insert Into Sorted Temp Table  ID = tn
  ```

- After a table access that contains a push-down operation to create a temporary table (that is, a create temporary table that occurs within the scope of a table access), there will be a ″completion″ statement, which handles end-of-file by getting the temporary table ready to provide rows to subsequent temporary table access. One of the following will be displayed:

  ```
  Temp Table Completion  ID = tn
  ```

  or

  ```
  Sorted Temp Table Completion  ID = tn
  ```

4.6.2.4 Access Table Queues

Table queues are used to communicate between subsections. A table queue is created only once. Repetitive insertions can occur on the same table queue. If a table queue is needed, the following statement is displayed in the explain report.

Access Table Queue: The text shows the table queue identifier and the number of columns in the table queue. The format is:

```
Access Table Queue ID = qn  #Columns = n
```

Access Table Queue Substatements: The following text can accompany the access table queue text in the explain report:

- `Output Sorted  #Columns = n`
- `Output Sorted and Unique  #Columns = n`
- `Sargable Predicate(s)`
- `Residual Predicate(s)`

The access table queue statement shows whether or not the rows of the table queue were sorted, and if so, how many columns of the table queue were used as the ordering key.

Create/Insert into Table Queue: This statement shows that rows will be inserted into a table queue and the table queue will be used to communicate between subsections. The text indicates the table queue number and the routing method used for the communication. The following are the possible routing methods:

- `Broadcast`
- `Directed`
- `Returned to sending node`
- `Directed to Selected node`

The format is:

```
Create/Insert Into Table Queue  ID = qn, xxxxxx
```

4.6.2.5 Joins

Four join strategies are implemented:

- Left Outer Merge Join
- Left Outer Nested Loop Join
- Merge join
- Nested loop join

Note: The query optimizer converts right outer joins into left outer joins by interchanging the left and right tables.

The outer table of the join will be the table referenced in the previous access statement shown in the output. The inner table of the join will be the table referenced in the access statement that is contained within the scope of the join statement. If a join involves more than two tables, the access statements should be read from top to bottom.

Join Substatements: The following text accompanies the join statement in the explain report to indicate the parallel join strategy chosen by the DB2 Parallel Edition optimizer. The possible join strategies are:

- Broadcast Outer Table

  The rows of the outer-table are transmitted to all nodes of the inner-table, where the join takes place.

- Directed Outer Table

  Each row of the outer-table is hashed based on the join columns corresponding to the inner-table partitioning attributes and sent to the appropriate nodes of the inner-table, where the join takes place.

- Directed Inner and Outer Tables

  The rows of the outer and inner-table are hashed based on their join columns and directed to common nodes, where the join takes place.

- Collocated

  The rows of the outer and inner-table are joined locally on each node.

4.6.2.6 Insert, Update, and Delete

The explain text for these SQL statements is self-explanatory. Possible statement text for these SQL operations can be:

- Insert With Values Clause: Table Name = qualifier.tbname ID = xx
- Insert With Select: Table Name = qualifier.tbname ID = xx
- Update: Table Name = qualifier.tbname ID = xx
- Update Current of Cursor: Table Name = qualifier.tbname ID = xx
- Delete: Table Name = qualifier.tbname ID = xx
- Delete Current of Cursor: Table Name = qualifier.tbname ID = xx

4.6.2.7 Index OR Filter

For some access plans, it is more efficient if the qualifying row identifiers (RIDs) are sorted and duplicates removed before the actual table access is performed:

```
Index OR Filter
```

Index ORing refers to the technique of making more than one index access and combining the results. The optimizer will consider index ORing when predicates are connected by OR keywords or there is an IN predicate. The index accesses can be on the same index or different indices.

4.6.2.8 Aggregation

Aggregation is performed on those rows meeting the specified criteria, if any, provided by the SQL statement predicates. If an aggregation function is to be done, one of the following statement appears:

```
Predicate Aggregation
Aggregation
```

Predicate aggregation states that the aggregation operation has been pushed-down to be processed as a predicate when the data is actually accessed.

Beneath the aggregation statement will be an indication of the type of aggregation function being performed:

```
Group By
Column Function(s)
Having
Single-fetch
```

The specific column function can be derived from the original SQL statement. A single record may be fetched from an index to satisfy a MIN or MAX operation.

If predicate aggregation is used, then subsequent to the table access statement in which the aggregation appeared, there will be an aggregation ″completion,″ which carries out any needed processing upon completion of each group or on end-of-file. The following line is displayed:

```
Aggregation Completion
```

The database manager does as much aggregation as it can as close to the source of data as possible. Completely local aggregation is possible if all partitioning columns are also grouping columns. If complete local aggregation is not possible, the database manager tries to create local partial groups, then merges the groups generated on the different nodes. Merging of the partial groups can occur at the consumer node or at the coordinator node. In either situation, during global aggregation the number of contributing rows to a single resulting row is less than or equal to the number of nodes where the local groups are produced, because each node can contribute, at most, one partial group for each grouping value. There are some cases where the DISTINCT clause may disallow even

partial local aggregation. These occur when the DISTINCT operator is applied to a column that is not the single-column partitioning key, and the grouping columns do not include all columns of the partitioning key.

4.6.2.9 Miscellaneous Statements

- Sections for data definition language statements will be indicated in the output with the following:

  ```
  DDL Statement
  ```

  No additional explain output is provided for DDL statements.

- Sections for LOCK TABLE statements will be indicated in the output with the following

  ```
  Lock Table
  ```

  No additional explain output is provided for LOCK TABLE statements.

- Sections for GRANT or REVOKE statements will be indicated in the output with one of the following:

  ```
  Authorization (Grant)
  Authorization (Revoke)
  ```

  No additional explain output is provided for GRANT or REVOKE statements.

- If the SQL statements contains the DISTINCT clause, the following text may appear in the output:

  ```
  Distinct Filter  #Columns = n
  ```

  where n is the number of columns involved in obtaining distinct rows. To retrieve distinct row values, the rows must be ordered so that duplicates can be skipped. This statement will not appear if the database manager does not have to explicitly eliminate duplicates, as in the following cases:

  - A unique index exists and all the columns in the index key are part of the DISTINCT operation
  - Duplicates that can be eliminated during sorting

- One of the following statements will appear if there is a set operator in the SQL statement:

  ```
  UNION
  UNION ALL
  EXCEPT
  EXCEPT ALL
  INTERSECT
  INTERSECT ALL
  ```

- One of the following statements will appear if an ALL, ANY, or EXISTS subquery is being processed in the access plan:

  ```
  ALL
  ANY
  ```

Note that the some IN predicates may be converted to equivalent quantified predicates:

- An IN predicate of the form:

```
expression IN (fullselect)
```

is equivalent to a quantified predicate of the form:

```
expression = ANY (fullselect)
```

- An IN predicate of the form:

```
expression NOT IN (fullselect)
```

is equivalent to a quantified predicate of the form:

```
expression <> ALL (fullselect)
```

4.7 Table Queues

Table queues are a mechanism used to communicate between processes which are cooperating in executing a query. They are implemented as data buffers, but to the database process the communication looks like a table access.

During communication there are always senders and receivers. Multiple different types of table queues exist where the number of senders and receivers differ. These are:

- Single receiver, single sender
- Single receiver, multiple sender
  - Non-deterministic interleaf
  - Deterministic interleaf (Merged table queue)
- Multiple receiver, single sender (Routed or Broadcast table queue)

4.7.1 Single-Receiver, Single-Sender Table Queues

This is the simplest case where one node transmits data to only one other node. Figure 41 illustrates this process.

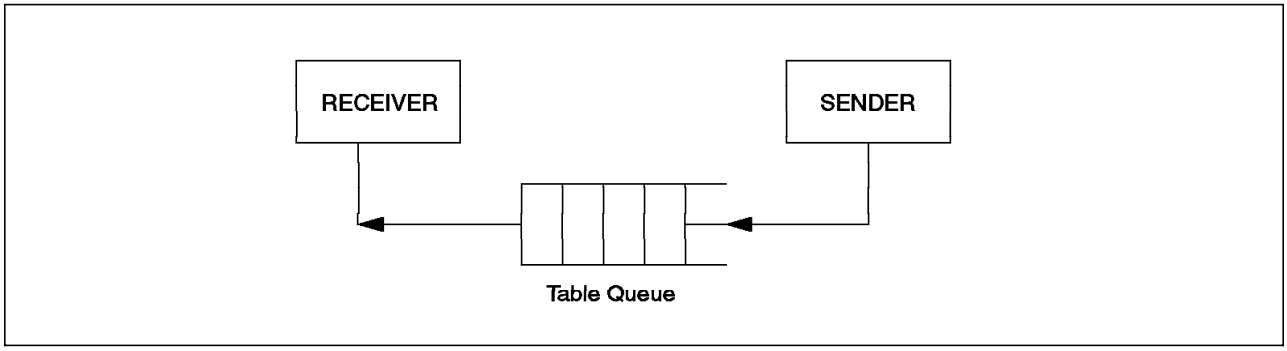


Figure 41. Single-Sender, Single-Receiver Table Queue

4.7.2 Single-Receiver, Multiple-Sender Table Queues

In this case multiple processes need to send data to one process. This may happen, for example, at the end of a select, where all the nodes involved in the query return their data to the coordinator node. This node would then return the collected output back to the application.

It may or may not be important to have a specific order in the received rows. If the order is not important then the table queue type will be a non-deterministic interleaf. This is illustrated in Figure 42.

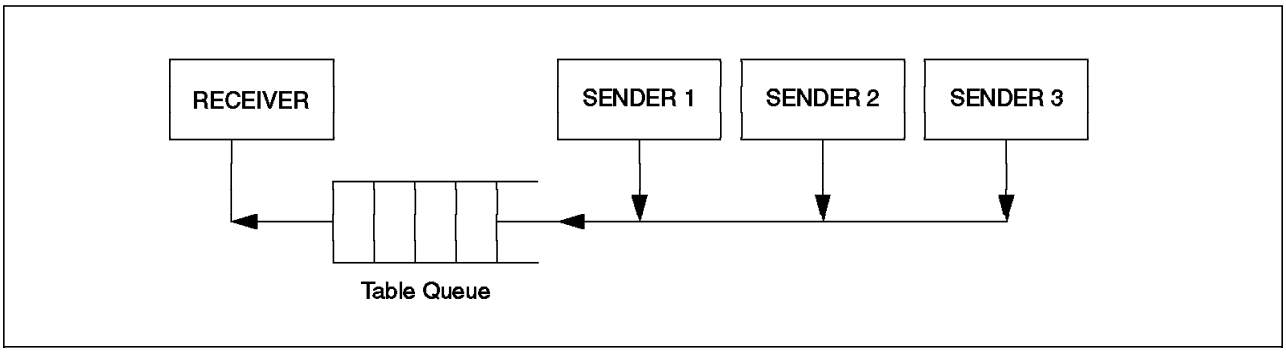


Figure 42. Non-Deterministic Interleaf Table Queue

If the order is important, then the merge of the data-streams must be controlled. This type of queue is called a deterministic interleaf. A deterministic interleaf is illustrated in Figure 43.

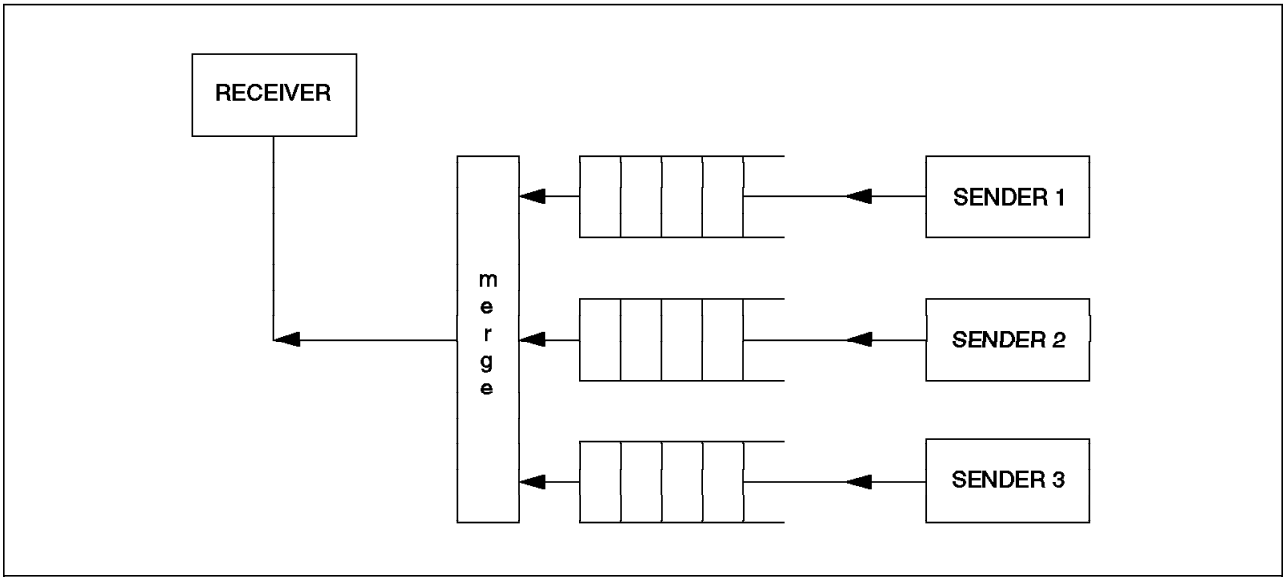


Figure 43. Deterministic Interleaf Table Queue

4.7.3 Multiple-Receiver, Single-Sender Table Queues

In this case one node sends a message to many of the nodes in the nodegroup. This case applies when, for example, the coordinator node splits a select and sends it to all nodes in the nodegroup. Two different kind of transmissions may occur:

Routed When column values are used to determine which receiver will receive which record

Broadcast When every record is sent to every receiver.

Figure 44 illustrates this type of queue.

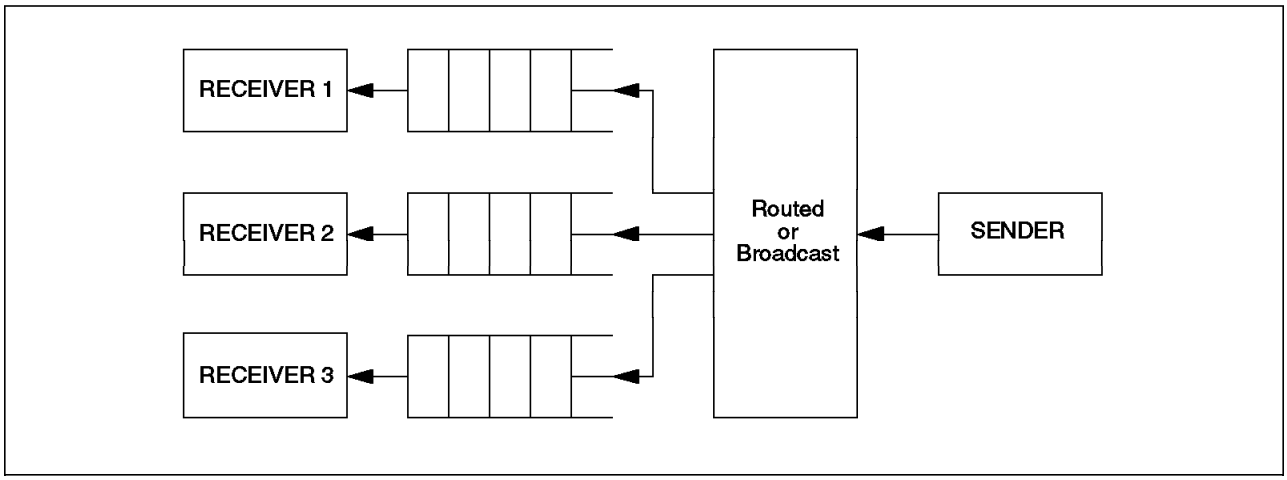


Figure 44. Single-Sender, Multiple-Receiver Table Queue

4.8 Join Operations

Join operations in DB2 Parallel Edition can be executed in parallel across many nodes. When generating a plan, the Parallel Edition optimizer considers different joining methods based on the partitioning keys and statistics information. Partitioning keys have a strong influence as to the type of join operations that the DB2 Parallel Edition optimizer will choose. Section 3.7.4, “Data Partitioning” on page 64 describes how to choose a good partitioning key.

4.8.1 Join Methods

DB2 Parallel Edition implements two join methods: Nested loop, and Merge scan.

4.8.1.1 Nested Loop Join Method

The Nested loop join repetitively scans the inner table. That is, DB2 scans the outer table once, and scans the inner table as many times as the number of qualifying rows in the outer table. Hence, the nested loop join is usually the most efficient join method when the values of the join column passed to the inner table are in sequence and the index on the join column of the inner table is clustered, or the number of rows retrieved in the inner table through the index is small.

The Nested loop join is often used if:

- The outer table is small.
- Predicates with small filter factors reduce the number of qualifying rows in the outer table.
- An efficient, highly clustered index exists on the join columns of the inner table.
- The number of data pages accessed in the inner table is small.

4.8.1.2 Merge Scan Join Method

For the Merge scan join DB2 scans both tables in the order of the join columns. If no efficient indices on the join columns provide the order, DB2 might sort the outer table, the inner table, or both. The inner table is always put into a work file; the outer table is put into a work file only if it must be sorted. When a row of the outer table matches a row of the inner table, DB2 returns the combined rows.

DB2 then reads another row of the inner table that might match the same row of the outer table and continues reading rows of the inner table as long as there is a match. When there is no longer a match, DB2 reads another row of the outer table.

- If that row has the same value in the join column, DB2 reads again the matching group of records from the inner table. Thus, a group of duplicate records in the inner table is scanned as many times as there are matching records in the outer table.
- If the outer row has a new value in the join column, DB2 searches ahead in the inner table. It can find:
  - Unmatched rows in the inner table, with lower values in the join column.
  - A new matching inner row. DB2 then starts the process again.
  - An inner row with a higher value of the join column. Now the row of the outer table is unmatched. DB2 searches ahead in the outer table, and can find:
    - Unmatched rows in the outer table.
    - A new matching outer row. DB2 then starts the process again.
    - An outer row with a higher value of the join column. Now the row of the inner table is unmatched, and DB2 resumes searching the inner table.

The Merge scan join is often used if:

- The qualifying rows of the inner and outer table are large, and the join predicate does not provide much filtering; that is, in a many-to-many join.
- The tables are large and have no indexes with matching columns.

- Few columns are selected on inner tables. This is the case when a DB2 sort is used. The fewer the columns to be sorted, the more efficient the sort.

For both merge-scan, and nested-loop join methods, there are four parallel join strategies available to Parallel Edition:

- Collocated Joins
- Directed Outer-Table Joins
- Directed Inner-Table and Outer-Table Joins
- Broadcast Outer-Table Joins

4.8.2 Parallel and Join Strategies

This section uses the same SQL query throughout on the same tables to demonstrate four types of join strategies based on the choice of partitioning keys defined on the tables and the size of the tables.

The example queries use:

```
SELECT O_ORDERPRIORITY, COUNT(DISTINCT O_ORDERKEY)
FROM ORDERS, LINEITEM
WHERE L_ORDERKEY = O_ORDERKEY AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY O_ORDERPRIORITY
```

The table definition for table ORDERS:

Column Name	Data Type
O_ORDERKEY	INTEGER
O_CUSTKEY	INTEGER
O_ORDERSTATUS	CHAR(1)
O_TOTALPRICE	FLOAT
O_ORDERDATE	DATE
O_ORDERPRIORITY	CHAR(15)
O_CLERK	CHAR(15)
O_SHIPPRIORITY	INTEGER
O_COMMENT	CHAR(49)

The table definition for table LINEITEM:

Column Name	Data Type
L_ORDERKEY	INTEGER
L_PARTKEY	INTEGER
L_SUPPKEY	INTEGER
L_LINENUMBER	INTEGER
L_QUANTITY	INTEGER
L_EXTENDEDPRICE	FLOAT
L_DISCOUNT	INTEGER
L_TAX	INTEGER

Column Name	Data Type
L_RETURNFLAG	CHAR(1)
L_LINESTATUS	CHAR(1)
L_SHIPDATE	DATE
L_COMMITDATE	DATE
L_RECEIPTDATE	DATE

4.8.3 Collocated Join Strategy

A collocated join is a join strategy which is performed locally on each node which contains relevant data. Communication is not required between the nodes except to return the answer set to the coordinator node. This join strategy will be chosen by the DB2 Parallel Edition optimizer if the following conditions are met:

- For tables residing in a single-node nodegroup
  - All tables residing in single-node nodegroups are collocated if the nodegroups are located on the same node.
- For tables residing in multinode nodegroups
  - The joined tables must reside in the same nodegroup.
  - The partitioning key for the joined tables must have the same number of columns.
  - Corresponding partitioning key columns must be partition compatible.
  - There must be equijoin predicates on all corresponding partitioning key columns of the joined tables.

For the example query (See 4.8.2, “Parallel and Join Strategies” on page 104), the following are the partitioning keys defined to illustrate the collocated join.

Table Name	Partitioning Key
ORDERS	O_ORDERKEY
LINEITEM	L_ORDERKEY

4.8.3.1 Explain Statement from the Collocated Join

The following is the SQL statement that was executed and the explain output it generated:

```
SQL Statement:

  SELECT O_ORDERPRIORITY, COUNT(DISTINCT O_ORDERKEY)
  FROM ORDERS, LINEITEM
  WHERE L_ORDERKEY=O_ORDERKEY AND L_COMMITDATE < L_RECEIPTDATE
  GROUP BY O_ORDERPRIORITY


(1)   Coordinator Subsection:
(2)     Distribute Subsection #1
(3)       Broadcast to Nodegroup
(4)       Partition Map ID = 4, Nodegroup = FOUR, #Nodes = 4
```

```
(5)     Access Table Queue  ID = q1  #Columns = 2
(6)       Output Sorted  #Columns: 2
(7)       Residual Predicate(s)
(8)         #Predicates = 1
(9)         Predicate Aggregation
(10)          Group By
(11)          Column Function(s)
(12)    Aggregation Completion
(13)      Group By
(14)      Column Function(s)

(15)  Subsection #1:
(16)    Access Table Name = DB2PE.LINEITEM  ID = 24  #Columns = 3
(17)      Scan Direction  = Forward
(18)      Relation Scan
(19)      Lock Intent Share
(20)      Sargable Predicate(s)
(21)        #Predicates = 2
(22)        Create/Insert Into Sorted Temp Table  ID = t1
(23)          Sort  #Columns = 1
(24)          Not Piped
(25)    Sorted Temp Table Completion  ID = t1
(26)    Access Table Name = DB2PE.ORDERS  ID = 23  #Columns = 2
(27)      Scan Direction  = Forward
(28)      Relation Scan
(29)      Lock Intent Share
(30)      Sargable Predicate(s)
(31)        #Predicates = 1
(32)        Create/Insert Into Sorted Temp Table  ID = t2
(33)          Sort  #Columns = 1
(34)          Piped
(35)    Sorted Temp Table Completion  ID = t2
(36)    Access Temp Table  ID = t2  #Columns = 2
(37)      Scan Direction  = Forward
(38)      Relation Scan
(39)    Merge Join
(40)      Join Strategy: Collocated
(41)      Access Temp Table  ID = t1  #Columns = 1
(42)        Scan Direction  = Forward
(43)        Relation Scan
(44)        Residual Predicate(s)
(45)          #Predicates = 1
(46)          Create/Insert Into Sorted Temp Table  ID = t3
(47)            Sort  #Columns = 2
(48)            Not Piped
(49)            Duplicate Reduction
(50)    Sorted Temp Table Completion  ID = t3
(51)    Access Temp Table  ID = t3  #Columns = 2
(52)      Scan Direction  = Forward
(53)      Relation Scan
(54)      Residual Predicate(s)
(55)        #Predicates = 1
(56)        Predicate Aggregation
(57)          Group By
(58)          Column Function(s)
(59)    Aggregation Completion
(60)      Group By
(61)      Column Function(s)
(62)    Create/Insert Into Table Queue  ID = q1, Broadcast
```

The following serves as an explanation of the explain statement that was shown in 4.8.3.1, “Explain Statement from the Collocated Join” on page 105 performing a collocated join.

- Coordinator subsection

  This subsection will be executed on the coordinator node where the application issues the CONNECT SQL statement.

  – In line **2** thru **4**

    - The coordinator subsection broadcasts subsection #1 to the 4 nodes of nodegroup FOUR.

  – In line **5** thru **14**

    - Access 2 columns (O_ORDERPRIORITY, and O_ORDERKEY) from table queue q1 which is created in line Line **51** thru **54**.

    - Finish applying the predicate for the GROUP BY and COUNT function.

    - Complete the aggregation and send the results back to user application.

- Subsection #1

  This subsection will be executed on the 4 nodes defined in nodegroup FOUR.

  – Line **16** thru **25**

    - Read 3 columns (L_ORDERKEY, L_COMMITDATE, and L_RECEIPTDATE) from table with FID = 24. The table name is DB2PE.LINEITEM.

    - Fetch the rows by scanning the table sequentially.

    - Apply 2 predicates. This creates a temporary table, t1, and inserts the rows which meet the criteria of the predicates to it. The temporary table t1 is sorted on L_ORDERKEY.

      Note: Because this temporary table will later be used as the inner table of a join, the sort cannot be piped.

    - Indicate the completion of sorted temporary table t1 creation.

  – Line **26** thru **35**

    - Read 2 columns (O_ORDERKEY, O_ORDERPRIORITY) from table with FID = 23. The table name is DB2PE.ORDERS.

    - Fetch the rows by scanning the table sequentially.

    - Apply 1 predicate. Create the temporary table t2 and insert the rows which meet the criteria of the predicate to it. Sort on O_ORDERKEY then pipe the sorted rows into the temporary table t2.

      Note: In a piped temporary table, the rows are sorted in memory and without creating a temporary table on disk, the sort pipes the rows out to join with the other table.

- Indicate the completion of sorted temporary table t4 creation.

– Line 36 thru 50
  - Outer-table (line 36 thru 38)
    - Access the temporary table t2 thru the pipe which is created in line 32.
    - Fetch the rows by scanning t2 sequentially.
  - The join method is merge join where the outer-table t4 and inner-table t1 are collocated. The join operation will be performed locally on each node.
  - Inner-table (line 41 thru 45)
    - Access the temporary table t1 created in line 22.
    - Fetch the rows by scanning t1 sequentially and apply 1 predicate.
  - Create the temporary table t3 and insert the rows which meet the criteria of the predicates to it. Sort the temporary table t3 in the order of O_ORDERPRIORITY, and O_ORDERKEY. Duplicate rows for the sorted temporary table are removed.
  - Indicate the completion of sorted temporary table t3 creation.

– Line 51 thru 62
  - Access the temporary table t3 sequentially.
  - Begin applying the predicate for the GROUP BY and COUNT function. Because the ORDERS table is partitioned on the O_ORDERKEY, the DISTINCT processing can be performed locally. The output from the aggregation will be a single row for each O_ORDERPRIORITY value.
  - Create and insert the rows from t3 to table queue q1. Broadcast the table queue q1.

4.8.3.2 Process Flow of a Collocated Join

Figure 45 on page 109 shows the processing and data transmission which will take place between two nodes, the coordinator node and another node.

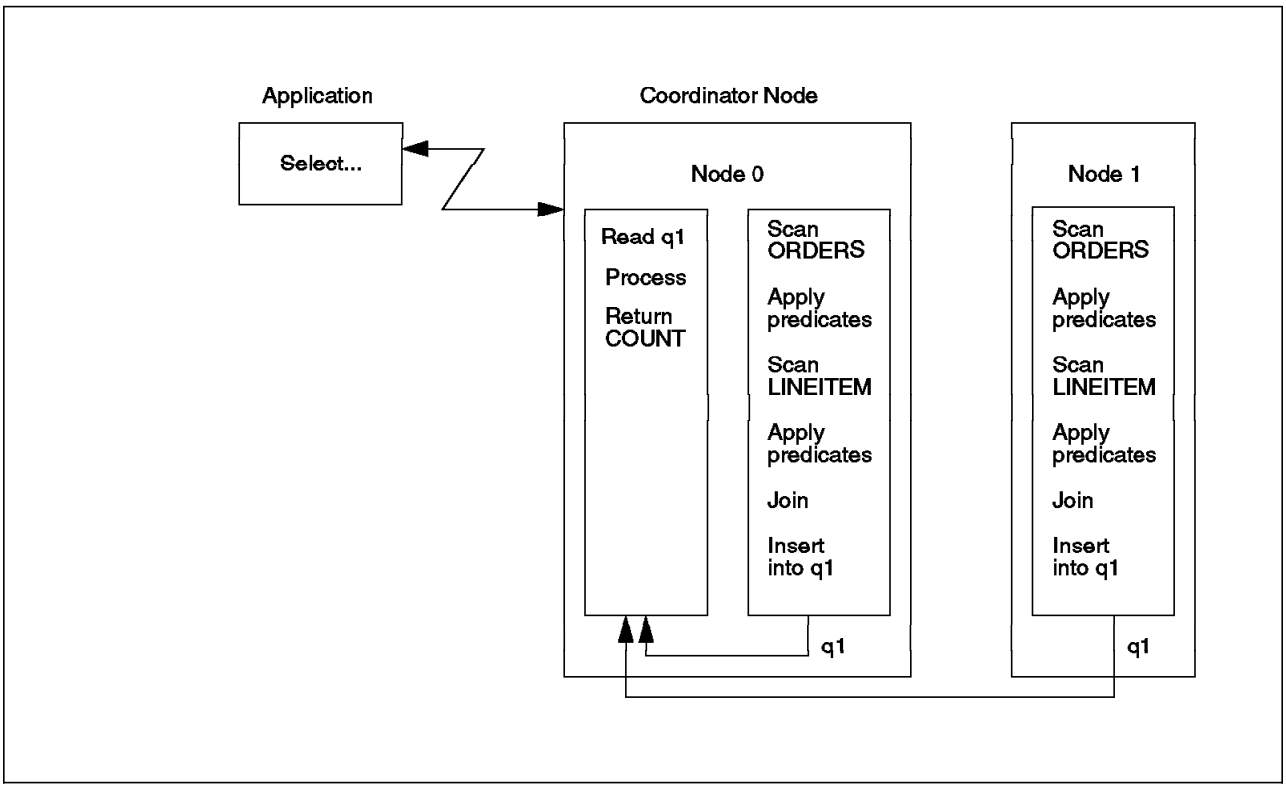


Figure 45. Collocated Join Process Flow

In Figure 45 the following process is illustrated:

1. The coordinator node receives a request from the application and sends it to all nodes containing relevant data.
2. The nodes scan the outer-table and may apply predicates to it.
3. The nodes scan the inner-table and may apply predicates to it.
4. The nodes perform the join operation between the two result sets.
5. The nodes send the results of the join to the coordinator node via table queue q1.
6. The coordinator node collects the results from table queue q1, processes them, and returns the final result to the user.

4.8.3.3 Data Flow of a Collocated Join

Figure 46 on page 110 shows the data transmission which occurs during the collocated join operation. Here the data is spread across four nodes and the coordinator node is shown as a separate node for clarity.

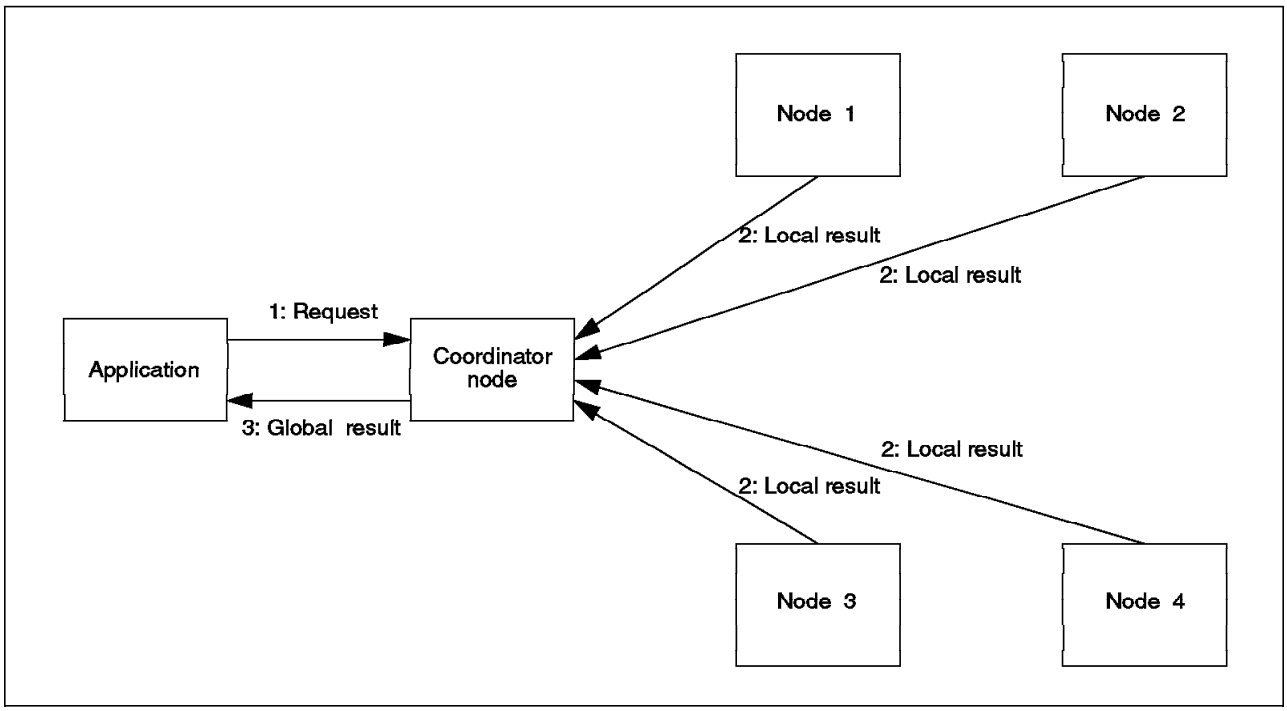


Figure 46. Collocated Join Data Flow

In Figure 46 the data flow is the following:

1. The application sends a request to the coordinator node. The coordinator node then splits the operation across all nodes containing relevant data (this is step 1 of the process flow).
2. The nodes send back their own results (this is step 5 of the process flow).
3. The coordinator node returns the final result to the application (this is step 6 of the process flow).

Step 1 requires little data transmission, while steps 2 and 3 may require the transmission of large amounts of data. This will depend on the answer set.

4.8.4 Directed Outer-Table Join Strategy

A directed outer join strategy may be chosen when the following conditions are satisfied:

- There must be equijoin predicates on all partitioning key columns of the inner table.

For the example query (See 4.8.2, “Parallel and Join Strategies” on page 104.), the following are the partitioning keys defined to illustrate the directed outer-table join:

Table Name	Partitioning Key
ORDERS	O_CUSTKEY
LINEITEM	L_ORDERKEY

4.8.4.1 Explain Statement for Directed Outer-Table Join

The following is the SQL statement that was executed and the explain output it generated:

```
SQL Statement:

  SELECT O_ORDERPRIORITY, COUNT(DISTINCT O_ORDERKEY)
  FROM ORDERS, LINEITEM
  WHERE L_ORDERKEY=O_ORDERKEY AND L_COMMITDATE < L_RECEIPTDATE
  GROUP BY O_ORDERPRIORITY


(1)   Coordinator Subsection:
(2)     Distribute Subsection #2
(3)       Broadcast to Nodegroup
(4)       Partition Map ID = 4, Nodegroup = FOUR, #Nodes = 4
(5)     Distribute Subsection #1
(6)       Broadcast to Nodegroup
(7)       Partition Map ID = 4, Nodegroup = FOUR, #Nodes = 4
(8)     Access Table Queue  ID = q1  #Columns = 2
(9)       Output Sorted  #Columns: 2
(10)      Residual Predicate(s)
(11)        #Predicates = 1
(12)        Predicate Aggregation
(13)          Group By
(14)          Column Function(s)
(15)    Aggregation Completion
(16)      Group By
(17)      Column Function(s)

(18)  Subsection #1:
(19)    Access Table Name = DB2PE.LINEITEM  ID = 24  #Columns = 3
(20)      Scan Direction  = Forward
(21)      Relation Scan
(22)      Lock Intent Share
(23)      Sargable Predicate(s)
(24)        #Predicates = 2
(25)        Create/Insert Into Sorted Temp Table  ID = t1
(26)          Sort  #Columns = 1
(27)          Not Piped
(28)    Sorted Temp Table Completion  ID = t1
(29)    Access Table Queue  ID = q2  #Columns = 2
(30)      Output Sorted  #Columns: 1
(31)    Merge Join
(32)      Join Strategy: Directed Outer Table
(33)      Access Temp Table  ID = t1  #Columns = 1
(34)        Scan Direction  = Forward
(35)        Relation Scan
(36)        Residual Predicate(s)
(37)          #Predicates = 1
(38)          Create/Insert Into Sorted Temp Table  ID = t2
(39)            Sort  #Columns = 2
(40)            Not Piped
(41)            Duplicate Reduction
(42)            Aggregation in Sort
(43)    Sorted Temp Table Completion  ID = t2
(44)    Access Temp Table  ID = t2  #Columns = 2
(45)      Scan Direction  = Forward
```

```
(46)      Relation Scan
(47)    Create/Insert Into Table Queue  ID = q1, Broadcast

(48)  Subsection #2:
(49)    Access Table Name = DB2PE.ORDERS  ID = 23  #Columns = 2
(50)      Scan Direction  = Forward
(51)      Relation Scan
(52)      Lock Intent Share
(53)      Sargable Predicate(s)
(54)        #Predicates = 1
(55)        Create/Insert Into Sorted Temp Table  ID = t3
(56)          Sort  #Columns = 1
(57)          Not Piped
(58)    Sorted Temp Table Completion  ID = t3
(59)    Access Temp Table  ID = t3  #Columns = 2
(60)      Scan Direction  = Forward
(61)      Relation Scan
(62)    Create/Insert Into Table Queue  ID = q2, Directed
```

The following serves as an explanation of the explain statement that was shown in 4.8.4.1, “Explain Statement for Directed Outer-Table Join” on page 111.

- Coordinator subsection

  This subsection will be executed on the coordinator node where the application issues the CONNECT SQL statement.

  - In line **2** thru **7**
    - The coordinator subsection broadcasts the subsection #1 and #2 to the 4 nodes of nodegroup FOUR.
  - In line **8** thru **17**
    - Access 2 columns (O_ORDERPRIORITY, and O_ORDERKEY) from table queue q1 which is created in line **47** .
    - Apply predicates for the GROUP BY and COUNT functions.
    - Complete the aggregation and send the results back to user application.

- Subsection #1

  This subsection will be executed on the 4 nodes defined in nodegroup FOUR.

  - Line **19** thru **28**
    - Read 3 columns (L_ORDERKEY, L_COMMITDATE, and L_RECEIPTDATE) from table with FID 24. The table name is DB2PE.LINEITEM.
    - Fetch the rows by scanning the table sequentially.
    - Apply 2 predicates. Create the temporary table t1 and insert the rows which meet the criteria of the predicates to it. Sort the temporary table t1 on 1 column.
    - Indicate the completion of sorted temporary table t1 creation.

- Line 29 thru 43
  - Outer-table (line 29 thru 30)
    - Access 2 columns from table queue q2 which is created in line 62. Table queue q2 is hashed and directed to 4 nodes defined in nodegroup FOUR. To make sure the received data from q2 remains sorted, table queue q2 performs deterministic interleaf to merge the rows.
  - The join method is merge join where the outer-table is directed.
  - Inner-table (line 33 thru 37)
    - Access 1 column from temporary table t1 which is created in line 25
    - Scan the temporary table t1 sequentially and apply 1 predicate.
  - Create the temporary table t2 and insert the rows which meet the criteria of the predicates to it. Sort the temporary table t2 in the order of O_ORDERPRIORITY, and O_ORDERKEY. Duplicate rows are removed, and the data is aggregated.
  - Indicate the completion of sorted temporary table t2 creation.
- Line 44 thru 47
  - Access 2 columns from temporary table t2 sequentially. t2 is created in line 38
  - Create and insert the rows from t2 to table queue q1. Broadcast the table queue q1.

- Subsection #2

  This subsection will be executed on the 4 nodes defined in nodegroup FOUR.

  - Line 49 thru 58
    - Read 2 columns (O_ORDERKEY, and O_ORDERPRIORITY) from table with FID 23. The table name is DB2PE.ORDERS.
    - Fetch the rows by scanning the table sequentially.
    - Apply 1 predicate. Create the temporary table t3 and insert the rows which meet the criteria of the predicates. Sort the temporary table t3 on O_ORDERKEY.
    - Indicate the completion of sorted temporary table t3 creation.
  - Line 59 thru 62
    - Access the temporary table t3 sequentially created in line 49 thru 58.
    - Create and insert the rows from t3 to table queue q2. Hash and direct the rows to the appropriate nodes via q2.

4.8.4.2 Process Flow for Directed Outer-Table Join

Figure 47 shows the processing and data transmission which will take place between two nodes: the coordinator node and another node. The operation is performed by hashing the rows of outer-table on the columns corresponding to the inner-table's partitioning key. Each outer table row is then directed to the target node generated from this, and a join operation performed on the target node.

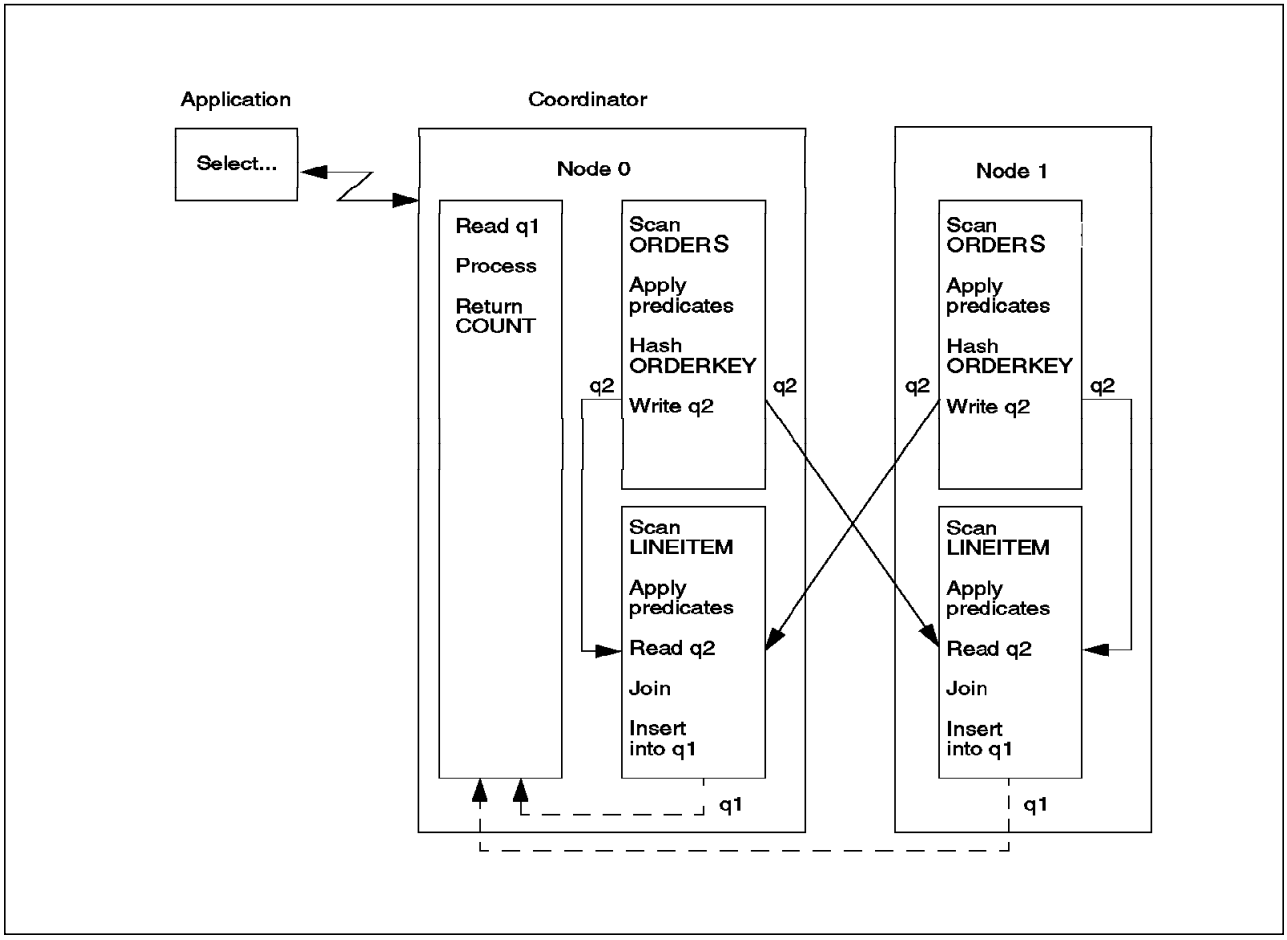


Figure 47. Directed Outer-Table Join Process Flow

In Figure 47 the following process is illustrated:

1. The coordinator node receives a request from the application and dispatches it to all nodes containing relevant data.
2. The nodes scan the outer-table and may apply predicates to it.
3. The nodes hash the outer-table using the join columns corresponding to the inner-table partitioning key.
4. The nodes send each outer-table row to the appropriate nodes determined from the hashing.
5. The nodes receive the outer-table rows.
6. The nodes scan the inner-table and may apply predicates to it.
7. The nodes perform the join operation between the inner-table and the received outer-table rows.
8. The nodes send the results of the join to the coordinator node.

9. The coordinator node collates these results, processes them and returns the final result to the application.

4.8.4.3 Data Flow for Directed Outer-Table Join

Figure 48 shows the data transmission which occurs during the directed outer table join operation. In this figure the data is spread across four nodes, and the coordinator node is shown as a separate node for clarity.

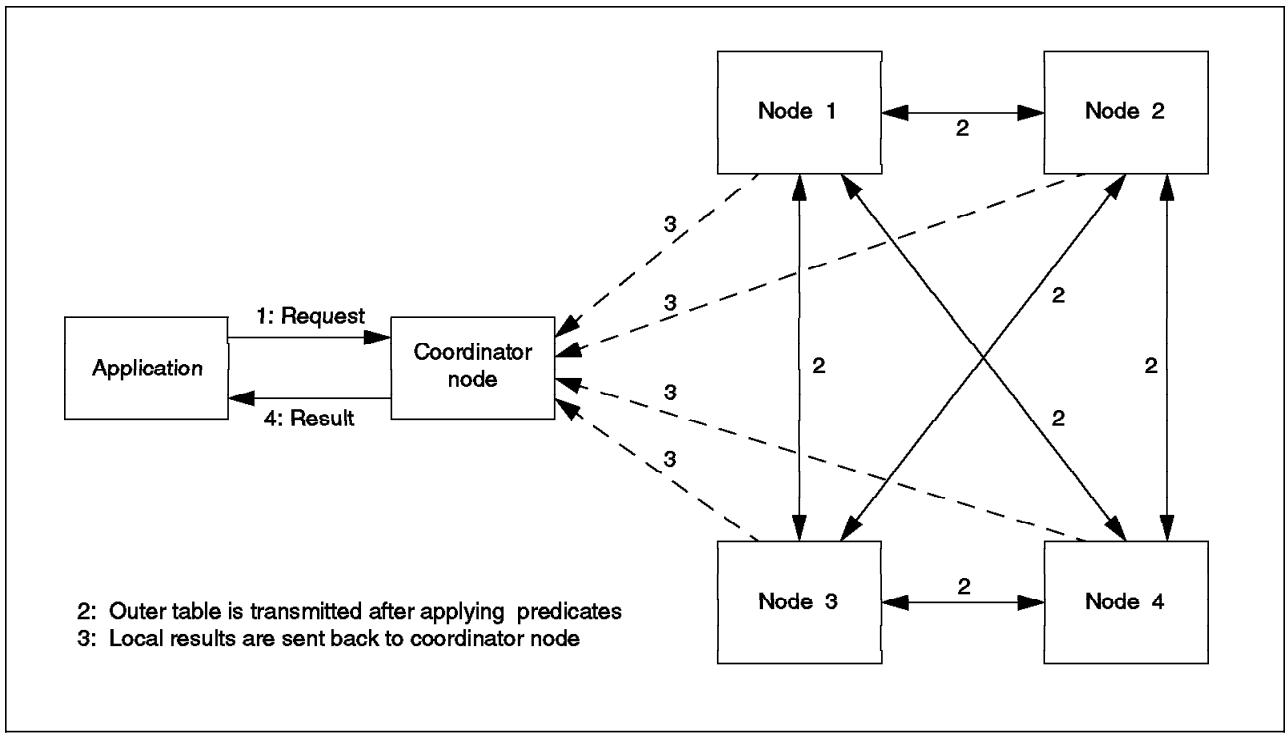


Figure 48. Directed Outer-Table Join Data Flow

In Figure 48 the data flow is the following:

1. The user send a request to the coordinator node which then splits this operation across nodes containing relevant data. (this is step 1 of the process flow).
2. The nodes direct the outer-table rows to the nodes containing the inner-table (this is steps 4 and 5 of the process flow).
3. The nodes send back their own results (this is step 8 of the process flow).
4. The coordinator node returns the final result to the application (this is step 9 of the process flow).

Step 1 involves little data transmission, while steps 2, 3, and 4 may require the transmission of a large amount of data. This is dependant on the operation requested.

4.8.5 Directed Inner-Table and Outer-Table Join Strategy

A directed inner-table and outer-table join strategy may be selected by DB2 Parallel Edition when both the inner table and the outer table join columns differ from their partitioning keys, but there is at least one equijoin predicate between the joined tables in the query.

For the sample query, the following is the partitioning keys defined to illustrate the directed inner-table and outer-table join:

Table Name	Partitioning Key
ORDERS	O_CUSTKEY
LINEITEM	L_PARTKEY

Note: The size of the ORDERS table is relatively large compared to LINEITEM table.

4.8.5.1 Explain Statement for Inner-Table and Outer-Table Join

The following is the SQL statement that was executed and the explain output it generated:

SQL Statement:

```
SELECT O_ORDERPRIORITY, COUNT(DISTINCT O_ORDERKEY)
FROM ORDERS, LINEITEM
WHERE L_ORDERKEY=O_ORDERKEY AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY O_ORDERPRIORITY
```

```
(1)   Coordinator Subsection:
(2)     Distribute Subsection #3
(3)       Broadcast to Nodegroup
(4)       Partition Map ID = 4, Nodegroup = FOUR, #Nodes = 4
(5)     Distribute Subsection #2
(6)       Broadcast to Nodegroup
(7)       Partition Map ID = 4, Nodegroup = FOUR, #Nodes = 4
(8)     Distribute Subsection #1
(9)       Broadcast to Nodegroup
(10)      Partition Map ID = 4, Nodegroup = FOUR, #Nodes = 4
(11)    Access Table Queue  ID = q1  #Columns = 2
(12)      Output Sorted  #Columns = 2
(13)      Residual Predicate(s)
(14)        #Predicates = 1
(15)        Predicate Aggregation
(16)          Group By
(17)          Column Function(s)
(18)    Aggregation Completion
(19)      Group By
(20)      Column Function(s)

(21)  Subsection #1:
(22)    Access Table Queue  ID = q3  #Columns = 1
(23)      Output Sorted #Columns = 1
(24)      Residual Predicate(s)
(25)        #Predicates = 1
(26)        Create/Insert Into Sorted Temp Table  ID = t1
(27)    Temp Table Completion  ID = t1
(28)    Access Table Queue  ID = q2  #Columns = 2
```

```
(29)      Output Sorted  #Columns = 1
(30)    Merge Join
(31)      Join Strategy: Directed Inner and Outer Table
(32)      Access Temp Table  ID = t1  #Columns = 1
(33)        Scan Direction  = Forward
(34)        Relation Scan
(35)        Residual Predicate(s)
(36)          #Predicates = 1
(37)         Create/Insert Into Sorted Temp Table  ID = t2
(38)            Sort  #Columns = 2
(39)            Piped
(40)            Duplicate Reduction
(41)    Sorted Temp Table Completion  ID = t2
(42)    Access Temp Table  ID = t2  #Columns = 2
(43)      Scan Direction  = Forward
(44)      Relation Scan
(45)      Residual Predicate(s)
(46)        #Predicates = 1
(47)        Predicate Aggregation
(48)          Group By
(49)          Column Function(s)
(50)    Aggregation Completion
(51)      Group By
(52)      Column Function(s)
(53)    Create/Insert Into Table Queue  ID = q1, Broadcast

(54)  Subsection #2:
(55)    Access Table Name = DB2PE.ORDERS  ID = 23  #Columns = 2
(56)      Scan Direction  = Forward
(57)      Relation Scan
(58)      Lock Intent Share
(59)      Sargable Predicate(s)
(60)        #Predicates = 1
(61)        Create/Insert Into Sorted Temp Table  ID = t3
(62)          Sort  #Columns = 1
(63)          Not Piped
(64)    Sorted Temp Table Completion  ID = t3
(65)    Access Temp Table  ID = t3  #Columns = 1
(66)      Scan Direction  = Forward
(67)      Relation Scan
(68)    Create/Insert Into Table Queue  ID = q2, Directed

(69)  Subsection #3:
(70)    Access Table Name = DB2PE.LINEITEM  ID = 24  #Columns = 3
(71)      Scan Direction  = Forward
(72)      Relation Scan
(73)      Lock Intent Share
(74)      Sargable Predicate(s)
(75)        #Predicates = 2
(76)        Create/Insert Into Sorted Temp Table  ID = t4
(77)          Sort  #Columns = 1
(78)          Not Piped
(79)    Sorted Temp Table Completion  ID = t4
(80)    Access Temp Table  ID = t4  #Columns = 1
(81)      Scan Direction  = Forward
(82)      Relation Scan
(83)    Create/Insert Into Table Queue  ID = q3, Directed
```

The following serves as an explanation of the explain statement that was shown in 4.8.5.1, “Explain Statement for Inner-Table and Outer-Table Join” on page 116.

- Coordinator subsection

  This subsection will be executed on the coordinator node where the application issues the CONNECT SQL statement.

  – In line **2** thru **10**

    - The coordinator subsection broadcasts subsections #1, #2, and #3 to the 4 nodes of nodegroup FOUR.

  – In line **11** thru **20**

    - Access 2 columns (O_ORDERPRIORITY, and O_ORDERKEY) from table queue q1 which is created in Line **53** .
    - Apply predicate for the GROUP BY and COUNT function.
    - Complete the aggregation and send the results back to user application.

- Subsection #1

  This subsection will be executed on the 4 nodes defined in nodegroup FOUR.

  – Line **22** thru **27**

    - Access 1 column from table queue q3 which is created in line **83** Table queue q3 is hashed and directed to 4 nodes defined in nodegroup FOUR.
    - Apply predicates. Creates the temporary table t1 and inserts the rows which meet the criteria of the predicates to it.
    - Complete inserting rows to the temporary table t1.

  – Line **28** thru **41**

    - Outer-table
      - Access 2 columns from table queue q2 which is created in line **68** Table queue q2 is hashed and directed to 4 nodes defined in nodegroup FOUR. To make sure the received data from q2 remains sorted, table queue q2 performs deterministic interleaf to merge the rows.
    - The join method is merge join where both the inner and outer-tables are directed.
    - Inner-table
      - Access 1 column from temporary table t1 which is created in line **26**
      - Scan the temporary table t1 sequentially and apply 1 predicate.
    - Create the temporary table t2 and insert the rows which meet the criteria of the predicates to it. Sort the temporary table t2. Remove duplicates for the rows in t2.

- Indicate the completion of sorted temporary table t2 creation.

– Line **42** thru **53**

- Access 2 columns from temporary table t2 sequentially. t2 is created in line **37**
- Apply predicate for the GROUP BY and COUNT function
- Complete the aggregation, create and insert the rows from t2 to table queue q1. Broadcast the table queue q1.

• Subsection #2

This subsection will be executed on the 4 nodes defined in nodegroup FOUR.

– Line **55** thru **64**

- Read 2 columns (O_ORDERKEY, and O_ORDERPRIORITY) from table with FID 23. The table name is DB2PE.ORDERS.
- Fetch the rows by scanning the table sequentially.
- Apply 1 predicates. Create the temporary table t3 and insert the rows which meet the criteria of the predicates to it. Sort the temporary table t3 on 1 column.
- Indicate the completion of sorted temporary table t3 creation.

– Line **65** thru **68**

- Access the temporary table t3 sequentially created in line **61** thru **64** .
- Create and insert the rows from t3 to table queue q2. Hash and direct the rows to the appropriate nodes via q2.

• Subsection #3

This subsection will be executed on the 4 nodes defined in nodegroup FOUR.

– Line **70** thru **79**

- Read 3 columns (L_ORDERKEY, L_COMMITDATE, and L_RECEIPTDATE) from table with FID 24. The table name is DB2PE.LINEITEM.
- Fetch the rows by scanning the table sequentially.
- Apply 2 predicates. Create the temporary table t4 and insert the rows which meet the criteria of the predicates to it. Sort the temporary table t4 on 1 column.
- Indicate the completion of sorted temporary table t4 creation.

– Line **80** thru **83**

- Access the temporary table t4 sequentially created in line **76** thru **79** .

- Create and insert the rows from t4 to table queue q3. Hash and direct the rows to the appropriated nodes via q3.

4.8.5.2 Process Flow for Inner-Table and Outer-Table Join

Figure 49 shows the processing and data transmission which will take place between two nodes, the coordinator node and another node. During a directed inner and outer table join, each qualifying row in either table is hashed on some join columns and then sent to the node specified by the partitioning map for the inner table of the join. Any predicates which can be applied before this transmission will be done to reduce the transmission volumes. The transmitted rows from the tables will then be joined locally on each of the nodes.

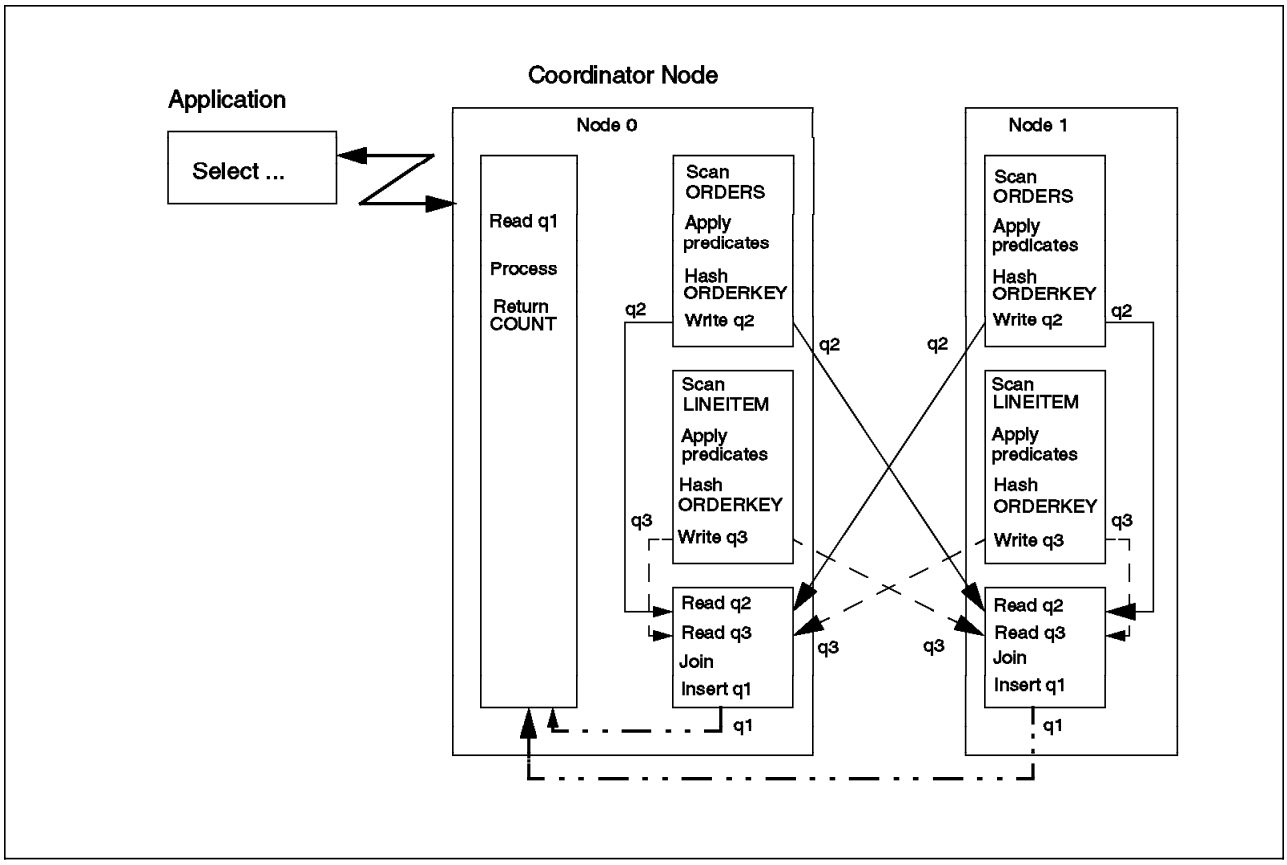


Figure 49. Directed Inner and Outer Join Process Flow

In Figure 49, the following process is illustrated:

1. The coordinator node receives the request from the application and dispatches it to all nodes.
2. Nodes containing the outer-table scan the outer-table and may apply predicates to it.
3. Nodes containing the inner-table scan the inner-table and may apply predicates to it.
4. Nodes containing the outer-table hash each row in the outer-table using the join columns.

5. Nodes containing the inner-table hash each row in the inner-table using the join columns.
6. The nodes direct these rows to the selected node.
7. Nodes receive the outer-table rows.
8. Nodes receive the inner-table rows.
9. Nodes perform the join operation between the received rows.
10. Nodes send the result of the join to the coordinator node.
11. The coordinator node collates these results, processes them and returns the final result to the application.

4.8.5.3 Data Flow for Inner-Table and Outer-Table Join

Figure 50 shows the data transmission which occurs during a directed inner and outer table join operation. In this figure the data is spread across four nodes, and the coordinator node is shown as a separate node for clarity.

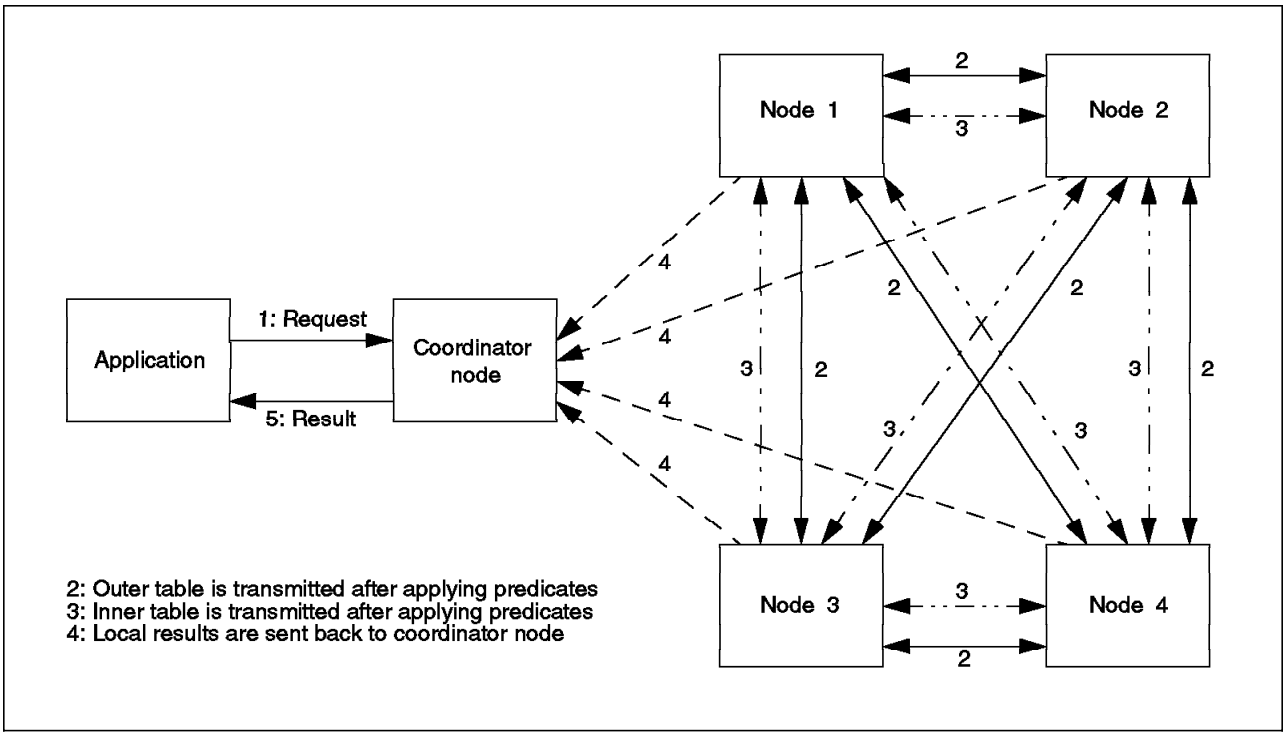


Figure 50. Directed Inner and Outer Join Data Flow

In Figure 50 the data flow is the following:

1. The application sends a request to the coordinator node. The coordinator node splits this operation across all nodes (this is step 1 of the process flow).
2. The nodes transmit rows of the outer-table to a node selected by hashing the join columns (this is steps 2, 4, and 6 of the process flow).
3. The nodes transmit rows of the inner-table to a node selected by hashing the join columns (this is steps 3, 5, and 6 of the process flow).

4. The nodes send back their own results (this is step 10 of the process flow).
5. The coordinator node returns the final result to the application (this is step 11 of the process flow).

More data transmission will be required for a directed inner and outer join than for the previous two types of joins. Step 1 will require little transmission of data, but now steps 2, 3, 4, and 5 may require the transmission of large amounts of data.

4.8.6 Broadcast Outer-Table Join Strategy

A broadcast outer-table join operation is always available to the optimizer. It will be chosen when none of the other join types is available, or sometimes if one of the tables is very small.

For the sample query, the following is the partitioning keys defined to illustrate the broadcast join:

Table Name	Partitioning Key
ORDERS	O_CUSTKEY
LINEITEM	L_PARTKEY

4.8.6.1 Explain Statement for Broadcast Outer-Table Join

The following is the SQL statement that was executed and the explain output it generated:

SQL Statement:

```
SELECT O_ORDERPRIORITY, COUNT(DISTINCT O_ORDERKEY)
FROM ORDERS, LINEITEM
WHERE L_ORDERKEY=O_ORDERKEY AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY O_ORDERPRIORITY
```

```
(1)   Coordinator Subsection:
(2)     Distribute Subsection #2
(3)       Broadcast to Nodegroup
(4)       Partition Map ID = 4, Nodegroup = FOUR, #Nodes = 4
(5)     Distribute Subsection #1
(6)       Broadcast to Nodegroup
(7)       Partition Map ID = 4, Nodegroup = FOUR, #Nodes = 4
(8)     Access Table Queue  ID = q1  #Columns = 2
(9)       Output Sorted  #Columns: 2
(10)      Residual Predicate(s)
(11)        #Predicates = 1
(12)        Predicate Aggregation
(13)          Group By
(14)          Column Function(s)
(15)    Aggregation Completion
(16)      Group By
(17)      Column Function(s)

(18)  Subsection #1:
(19)    Access Table Queue  ID = q2  #Columns = 2
(20)      Output Sorted  #Columns = 2
(21)    Nested Loop Join
```

```
(22)      Join Strategy: Broadcast Outer Table
(23)      Access Table Name = DB2PE.LINEITEM  ID = 24  #Columns = 3
(24)        Scan Direction  = Forward
(25)        Relation Scan
(26)        Lock Intent Share
(27)        Sargable Predicate(s)
(28)          #Predicates = 3
(29)          Create/Insert Into Temp Table  ID = t1
(30)     Temp Table Completion  ID = t1
(31)     Access Temp Table  ID = t1  #Columns = 2
(32)       Scan Direction  = Forward
(33)       Relation Scan
(34)     Create/Insert Into Table Queue  ID = q1, Broadcast

(35)  Subsection #2:
(36)    Access Table Name = DB2PE.ORDERS  ID = 23  #Columns = 2
(37)      Scan Direction  = Forward
(38)      Relation Scan
(39)      Lock Intent Share
(40)      Sargable Predicate(s)
(41)        #Predicates = 1
(42)        Create/Insert Into Sorted Temp Table  ID = t2
(43)          Sort  #Columns = 2
(44)          Not Piped
(45)          Duplicate Reduction
(46)    Sorted Temp Table Completion  ID = t2
(47)    Access Temp Table  ID = t2  #Columns = 2
(48)      Scan Direction  = Forward
(49)      Relation Scan
(50)    Create/Insert Into Table Queue  ID = q2, Broadcast
```

The following serves as an explanation of the explain statement that was shown in 4.8.6.1, “Explain Statement for Broadcast Outer-Table Join” on page 122.

- Coordinator subsection

  This subsection will be executed on the coordinator node where the application issues the CONNECT SQL statement.

  – In line **2** thru **7**

    - The coordinator subsection broadcasts the subsection #1 and #2 to a nodegroup FOUR with 4 nodes defined.

  – In line **8** thru **17**

    - Access 2 columns (O_ORDERPRIORITY, and O_ORDERKEY) from table queue q1 which is created in line **29**.
    - Apply predicates for the GROUP BY and COUNT functions.
    - Complete the aggregation and send the results back to user application.

- Subsection #1

  This subsection will be executed on the 4 nodes defined in nodegroup FOUR.

  – Line **19** thru **34**

- Outer-table
  - Access 2 columns from table queue q2 which is created in line 50. Table queue q2 is broadcast to 4 nodes defined in nodegroup FOUR. To make sure the received data from q2 remains sorted, table queue q2 performs a deterministic merge.
- The join method is a nested loop join where the outer-table is broadcast.
- Inner-table
  - Read three columns (L_ORDERKEY, L_COMMITDATE, and L_RECEIPTDATE) from table with FID = 24. The table name is DB2PE.LINEITEM.
  - Fetch the rows by scanning the table sequentially.
  - Apply 2 predicates. Create the table queue q1 and inserts the rows which meet the criteria of the predicates to it.
  - Broadcast the table queue q1.

• Subsection #2

This subsection will be executed on the 4 nodes defined in nodegroup FOUR.

– Line 36 thru 46
  - Read two columns (O_ORDERKEY, and O_ORDERPRIORITY) from table with FID = 23. The table name is DB2PE.ORDERS.
  - Fetch the rows by scanning the table sequentially.
  - Apply one predicate. Create the temporary table t2 and insert the rows which meet the criteria of the predicates to it. Sort the temporary table t1. Remove duplicate rows in t2.
  - Indicate the completion of sorted temporary table t2 creation.

– Line 47 thru 50
  - Access the temporary table t2 sequentially created in line 42.
  - Create and insert the rows from t2 to table queue q2. Broadcast the data to 4 nodes defined in nodegroup FOUR via q2.

4.8.6.2 Process Flow for Broadcast Outer-Table Join

Figure 51 on page 125 shows the processing and data transmission which will take place between two nodes, the coordinator node and another node.

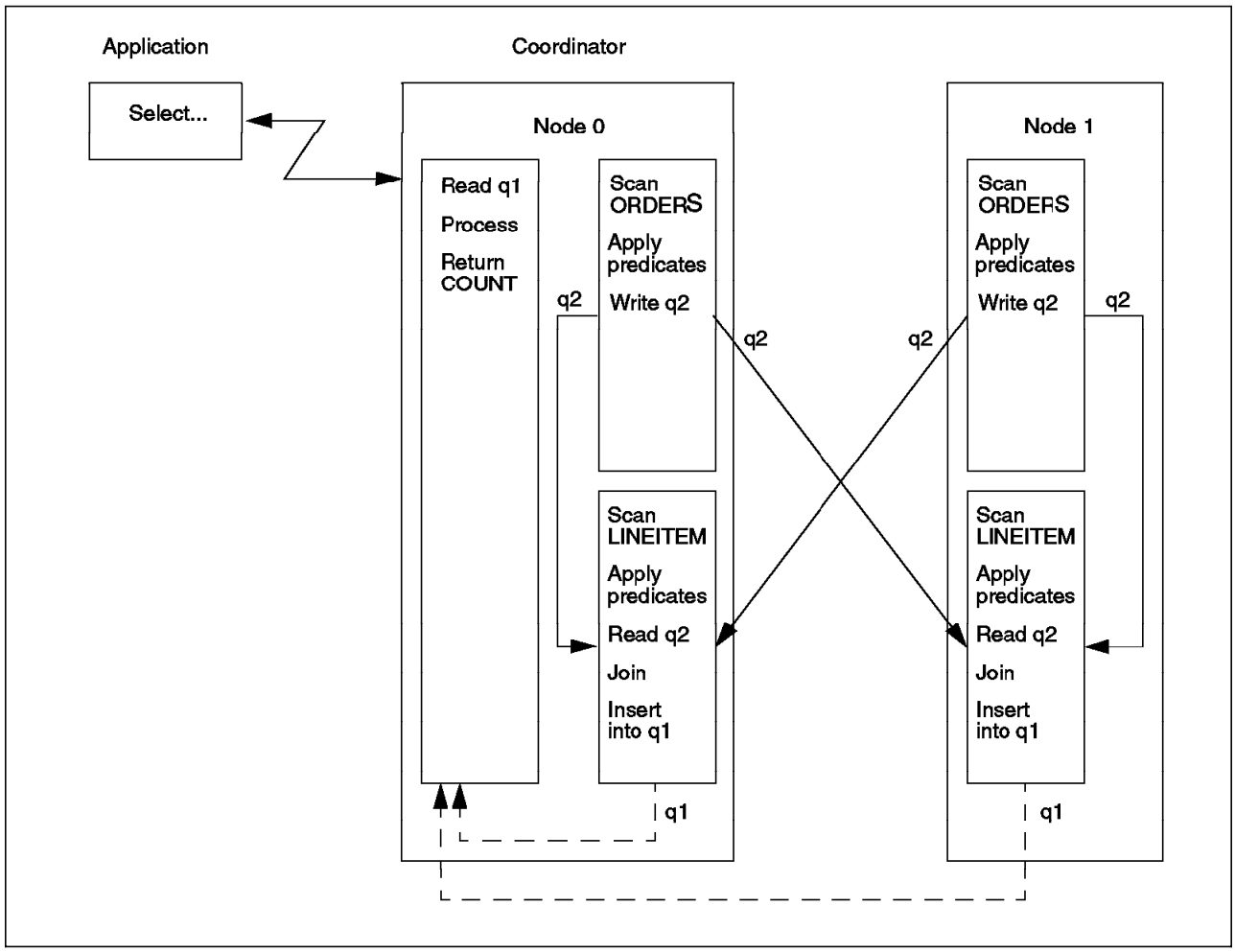


Figure 51. Broadcast Join Process Flow

In Figure 51, the following process is illustrated:

1. The coordinator node receives the request from the user and dispatches it to all nodes containing relevant data.
2. The nodes scan the outer-table and may apply predicates to it.
3. The nodes transmit the full resultant outer-table to all the nodes.
4. The nodes receive the outer-table.
5. The nodes scan the inner-table and may apply predicates to it.
6. The nodes perform the join operation between the local inner-table and the received outer-table.
7. The nodes send the results of the join to the coordinator node.
8. The coordinator node collects the results, process them and returns the final result to the application.

4.8.6.3 Data Flow for Broadcast Outer-Table Join

Figure 52 shows the data transmission which occurs during a broadcast join operation. In this figure the data is spread across four nodes, and the coordinator node is shown as a separate node for clarity.

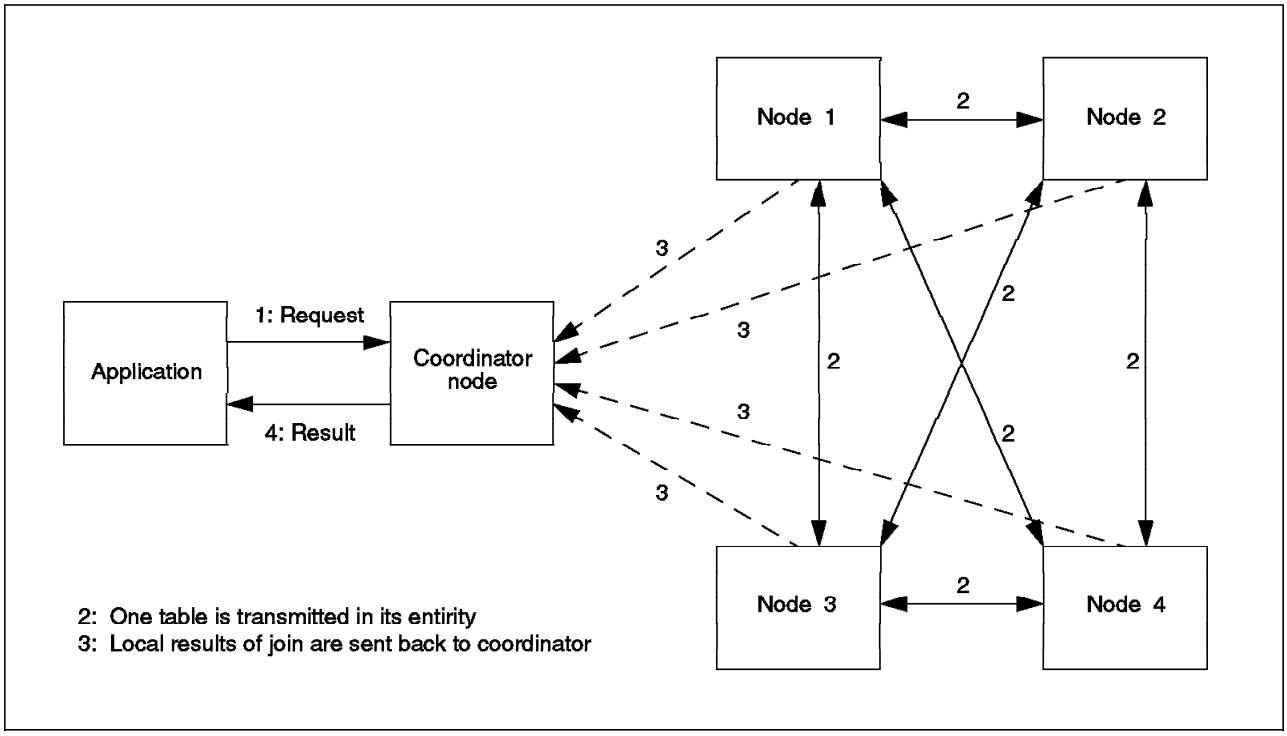


Figure 52. Broadcast Join Data Flow

In Figure 52, the data flow is the following:

1. The application sends a request to the coordinator node. The coordinator node then splits this operation across all nodes containing relevant data (this is step 1 of the process flow).
2. The nodes transmit the outer-table to all nodes (this is steps 3 and 4 of the process flow).
3. The nodes send back their own results (this is step 7 of the process flow).
4. The coordinator node returns the final result to the application (this is step 8 of the process flow).

Step 1 requires little data transmission, while steps 2, 3, and 4 may require the transmission of large amounts of data.

4.9 Database Locking

This section talks about the concept of locking in a database environment. The section is outlined as follows:

- Lock Modes
- Lock Mode Compatibility
- Lock Duration

- The lock table SQL statement
- Lock Conversion
- Lock Escalation
- Deadlock
- Distributed Deadlock Detection
- Concurrency Configuration Parameters

Locking is a database mechanism used to regulate concurrent processing of data. Concurrent processing means that multiple processes or applications can access the same database at the same time. Locking is used in DB2 Parallel Edition as in DB2/6000 Version 1 to maintain data integrity during concurrent processing. Locking ensures that a transaction maintains control over a database row until it has finished, and prevents another application from changing a row before the change in progress completes. Locks are used in DB2 PE to:

- Prevent loss of data that could occur in a simultaneous update.
- Prevent access to data whose status is in-doubt.
- Allow data that has been read by an application to be protected from change if so desired.

4.9.1 Lock Modes

In general, there are two basic types or modes of locks:

- Exclusive or X locks - These allow no other applications to change the resource they are locking. The one exception is an uncommitted read. The type of statements that will take these locks are update, insert and delete.
- Shared or S locks - These allow other applications to read the resource they are locking. They do not allow other application to change that resource. The SQL select statement will take these locks.

  DB2 PE provides locks on both the table level and the row level. The default is row level locking. Row level locking provides more control and concurrency over objects. However, row level locking requires more overhead. Indices also are subject to locks. Indices are locked through the use of row level locks that correspond to the index entry. Sometimes, internal locks are placed on indices. The database manager will choose which level of lock to take, unless the lock table SQL statement is issued. (See 4.9.3.2, “Strict Table Locking” on page 134 for more information.) Table 4 on page 128 and Table 5 on page 129 show the different lock modes available at the table and row levels.

4.9.1.1 Table-Level Locks

In addition to the basic types of locks, shared and exclusive, a number of other types of locks known as intention or intent locks are available. They offer more granularity to objects being locked. If an object is locked in an intention lock, it implies that explicit locking is being done at some lower level or at a finer granularity. Table 4 shows the locks that are permitted at the table level, including the intent locks.

Table 4. Lock Modes for DB2 PE - Table Level

Lock Modes	Definitions
IN	Intent None
IS•	Intent Share
S•	Share
IX•	Intent eXclusive
U•	Update
S•IX•	Share with Intent eXclusive
X•	eXclusive
Z•	Superexclusive

Note:

1. Row locking is also used.
2. Strict table locking.

The lock modes outlined in Table 4 are defined below. The modes are listed in order of their increasing control over resources:

- IN - Intent None

  The lock owner can read any data in the table, including uncommitted data, but cannot change any of it. No row locks are acquired by the lock owner. Other concurrent applications can read or update the table.

- IS - Intent Share

  The lock owner can read data in the locked table, but not change this data. When an application holds the IS table lock, the application acquires an S lock on each row read. In either case, other applications can read or update the table.

- Share - Share

  The lock owner can read, but not change data in the table. No row locks are acquired by the lock owner. Other concurrent applications can read the table.

- IX - Intent Exclusive

  The lock owner can read or change data in the table. When the owner reads data, it acquires an S, U, or X lock on each row. It also acquires an X lock on each row that it updates. and a U or S lock can be obtained on rows to be read. Other concurrent applications can both read and update the table.

- U - Update

The lock owner can change data in the table and acquires X locks on the rows prior to updates. Other applications can read the data, but cannot attempt to update it.

- SIX - Share with Intent Exclusive

  The lock owner can read or change data in the table. The lock owner acquires X locks on the rows it updates, but does not acquire locks on rows that it reads. Other concurrent applications can read the table.

- X - Exclusive

  The lock owner can read and change. No row locks are acquired by the lock owner. Only uncommitted read applications can access the locked table.

- Z - Super Exclusive

  The lock owner is supporting a `create table`, `drop table`, `alter table`, `create index`, or `drop index` statement on the table. No other applications can access the table, including uncommitted read applications.

4.9.1.2 Row-Level Locks

Table 4 on page 128 shows the locks available at the table level. Row locks are only requested by applications that have supporting locks at the table level. These supporting locks are the intent locks: IS, IX and SIX.

Table 5 shows the locks that are permitted at the row level.

Table 5. Lock Modes for DB2 PE - Row Level

Row Lock		Minimum• Supporting Table Lock
NS	Next key Share	IS
S	Share	IS
NX	Next key eXclusive	IX
U	Update	IX
X	eXclusive	IX

Note:

1. This denotes the least restrictive lock necessary. However, this does not imply that the table lock listed is the only table lock that supports the row lock listed. For example, an application that possesses an IX table lock could possess S, U or X locks on rows. Likewise, an application that possesses an SIX table lock could possess X locks on rows.

The definitions for row locks are similar to the definitions for corresponding table locks, except that the object of the lock is a row. The modes are listed in order of their increasing control over resources:

- NS - Next key Share

  The lock owner and any concurrent applications can read, but not change the locked row. This lock is acquired only on rows of the system catalog tables, instead of an S lock, by internal system catalog table scans.

- NX - Next key eXclusive

  The lock owner can read, but not change the locked row. This lock is acquired only on the next index key during insert or delete operations. Uncommitted read applications, and internal system catalog table scans usings NS row locks can read the locked row.

- S - Share

  The lock owner and any concurrent applications can read, but not change the locked row.

- U - Update

  The lock owner can change data in the locked row and acquires X locks on the rows prior to updates. Other applications can read the row, but cannot attempt to update it. The major difference between the U lock and the S lock is the intent to update. Only one application can possess a U lock on a row.

- X - Exclusive

  The lock owner can read and change data in the locked row. Only uncommitted read applications can read the locked row.

4.9.2 Lock Mode Compatibility

Compatibility between lock types and row locks also depends on the objects trying to access the data. Once an intent lock is obtained on a table, the corresponding row lock that goes with the table lock is automatically allowed. For example, a table with an intent lock of IS will allow row locks of S, but not X. Tables Table 6 and Table 7 assumes two applications exist, Application 1 (Lock 1) and Application 2 (Lock 2). The tables can be used to determine if the two applications can run concurrently if they are requesting access to the same table with a given lock mode.

Table 6. Lock Mode Compatibility - Table Locks

MODE OF LOCK 1	MODE OF LOCK 2							
	IN	IS	S	IX	SIX	U	X	Z
IN	YES	YES	YES	YES	YES	YES	YES	NO
IS	YES	YES	YES	YES	YES	YES	NO	NO
S	YES	YES	YES	NO	NO	YES	NO	NO
IX	YES	YES	NO	YES	NO	NO	NO	NO
SIX	YES	YES	NO	NO	NO	NO	NO	NO
U	YES	YES	YES	NO	NO	NO	NO	NO
X	YES	NO	NO	NO	NO	NO	NO	NO
Z	NO	NO	NO	NO	NO	NO	NO	NO

Table 7 (Page 1 of 2). Lock Mode Compatibility - Row Locks

MODE OF LOCK 1	MODE OF LOCK 2				
	NS	S	U	NX	X
NS	YES	YES	YES	YES	NO
S	YES	YES	YES	NO	NO

Table 7 (Page 2 of 2). Lock Mode Compatibility - Row Locks					
U	YES	YES	NO	NO	NO
NX	YES	NO	NO	NO	NO
X	NO	NO	NO	NO	NO

For example, consider the case when Application 1 obtains an IS lock against a given table. Application 2 could obtain an IN, IS, S, IX, SIX or U lock against the same table at the same time. However, an X or Z lock would not be allowed.

Many different applications could have compatible locks on the same object. For example, ten transactions may have IS locks on a table and five different transactions may have IX locks on the same table. There is no concurrency problem at the table level in such a scenario. However, there may be lock contention at the row level:

- The basic concept of Table 7 on page 130 is that rows being read by an application can be read by other applications and that rows being changed by an application are not available to other applications that use row locking.
- U level row locks are not compatible with other U level row locks. Only one application can read a row with the intent to update. The U lock reduces the number of deadlocks that occur when applications perform updates and deletes using cursors. When a row is fetched using a cursor declared FOR UPDATE OF, the U row lock is used.

4.9.3 Lock Duration

Concurrent access to the database is allowed via three different isolation levels that ensure data integrity through locking. The isolation level selected for an application can impact both the lock strategy and the duration of row locks. The isolation level can be specified during program preparation or bind. The default level is cursor stability. The three levels of isolation are:

- Repeatable Read (RR)
- Cursor Stability (CS)
- Uncommitted Read (UR)

Figure 53 on page 132 shows a table, Table 1, and the difference between the isolation levels.

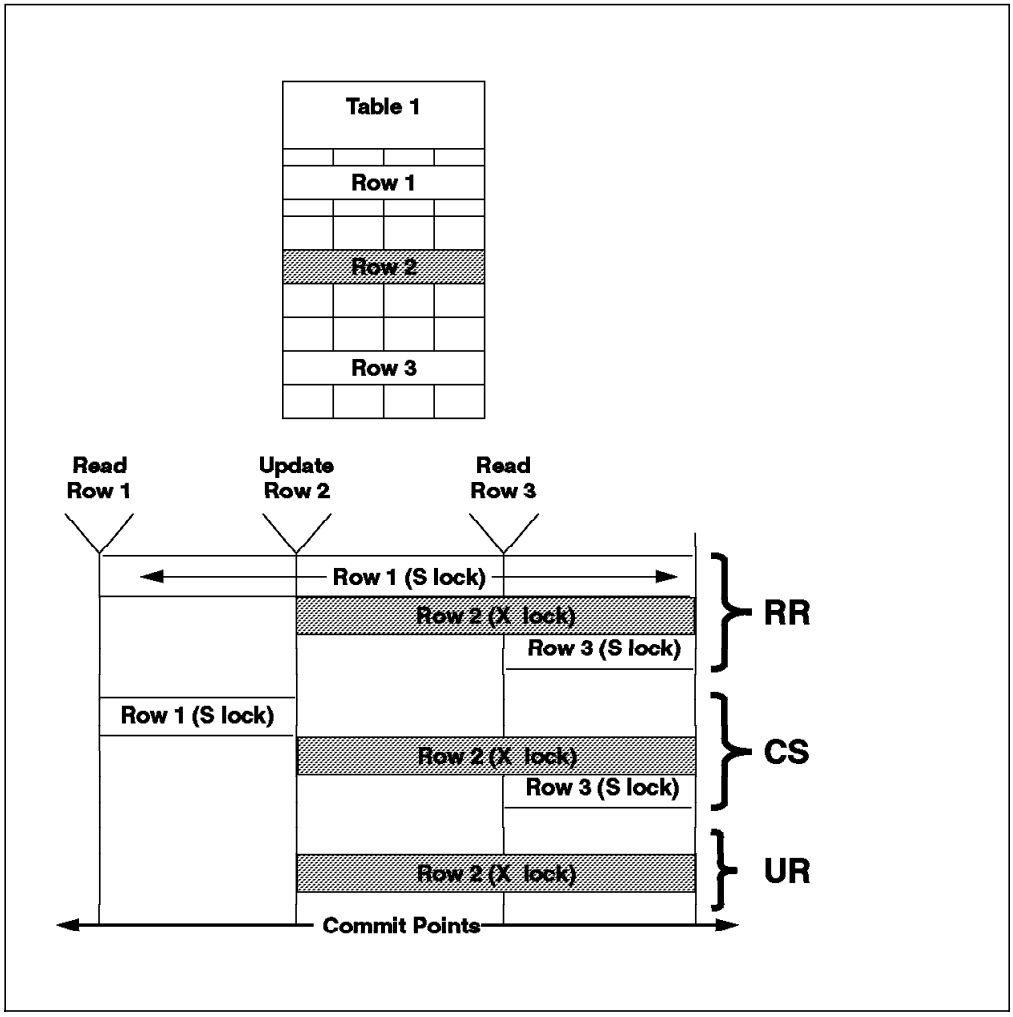


Figure 53. Isolation Levels Within DB2 PE

Figure 53 details the impact of isolation levels on row lock duration. There are other factors that influence lock strategy, such as temporary tables and access path. For Table 1 in Figure 53, assume that row locks are being obtained. The isolation levels are defined with respect to the diagram:

- Repeatable Read (RR)

  S row or table locks are held until the next commit or rollback occurs. If a row is read multiple times within a unit of work, the value for the row will not change.

  This isolation level may reduce concurrency at the row level, but may be useful for applications that require examination of multiple rows before a processing decision can be made.

- Cursor Stability (CS)

  S row locks are held only while the cursor is positioned on the row. Rows that have been previously examined within a unit of work are not locked.

  This isolation level increases concurrency at the row level, and is normally preferred for most application requirements.

However, Repeatable Read applications with table locks may out perform Cursor Stability applications due to reduced lock manager overhead.

- Uncommitted Read (UR)

  S row locks are not obtained.

  This isolation level may improve concurrency of applications since having to wait for S row locks is eliminated. However, the application programmer must be aware of any data integrity risks associated with this isolation level. There may also be performance improvements associated with the elimination of the lock manager overhead.

4.9.3.1 Impact of Temporary Tables on Lock Duration

DB2 PE may create a temporary table on a node to satisfy an application's requirements. For example, an order by clause specification that is not satisfied using an index access will cause the database manager to create a temporary table. Figure 54 shows a fetch from the temporary table rows of a base table are not locked if the isolation level is cursor stability (CS) or uncommitted read (UR).

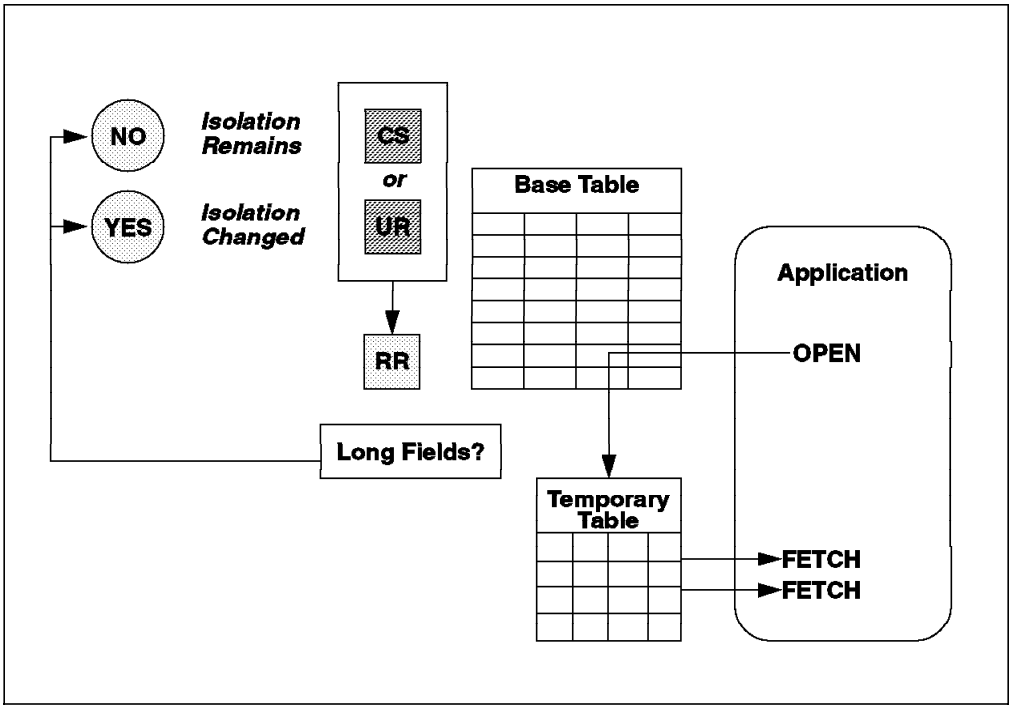


Figure 54. Temporary Tables and Impact on Isolation Level

Figure 54 shows what happens if the temporary table accesses long field data. Here, the isolation level on the base table will be changed from cursor stability (CS) or uncommitted read (UR) to repeatable read (RR). A corresponding decrease in concurrency may be experienced.

If the temporary table is created and long field data is not accessed, the prior isolation level on the base table is maintained. However, the S row locks for Cursor Stability applications are obtained and released during the OPEN of the cursor. The FETCH from the

temporary table may therefore present a row to the application that has since been changed in the underlying base table.

4.9.3.2 Strict Table Locking

Unless specifically requested by the application, the database manager will usually use both table and row lock strategies. However, there are cases when the database manager determines that such strategies would not be appropriate. A list of such strategies is as follows:

- Update/Delete that does not specify the WHERE clause
- Repeatable Read with a table scan
- Repeatable read with an index scan and no WHERE clause
- Create, alter or drop of index or table
- Lock Escalation
- The `lock table` SQL statement

The database manager will use table and row locking to satisfy an application's request to update or delete data through the use of an index. However, if the update or delete does not specify the WHERE clause, the database manager will use strict table locking.

If the situations involving Repeatable Read lead to unacceptable concurrency problems, the application should be examined to determine if a different access strategy can be used or if the isolation level can be changed. Repeatable Read can be logically simulated, although the application code required to do so may carry a high development and/or maintenance cost.

Strict table locking cannot be avoided when issuing DDL against a table or index. When possible, the database administrator should restrict these type of statements to periods of low user activity. Lock escalation occurs when there are too many row level locks. Lock escalation is covered in 4.9.5, "Lock Escalation" on page 135. Strict table locking can be requested by an application through the use of the `lock table` SQL statement.

The `lock table` SQL statement provides the application programmer with the flexibility to lock a table at a more restrictive mode than requested by the database manager. For example:

- `lock table tablename IN SHARE MODE`

  This allows other applications to read but not change any rows in the table

- `lock table tablename IN EXCLUSIVE MODE`

  This allows no other application to read or change any rows in the table unless they are uncommitted reads.

Locks obtained with the `lock table` SQL statement are acquired when the statement is executed. These locks are released by a commit or rollback. For the complete syntax of the `lock table` statement, see the *SQL Reference*.

4.9.4 Lock Conversion

Lock conversion occurs when an application requires a more restrictive lock on a resource than the one(s) it already holds as SQL statements are processed within a unit of work. For example, an application could convert from a shared lock to an exclusive lock.

The SIX table lock is a special case. It is obtained if an application possesses an IX lock on a table and requests an S lock, or vice versa. The result of lock conversion in these cases is the SIX lock.

4.9.5 Lock Escalation

Lock escalation occurs when there are too many row level locks. When this limit is reached, row locks are replaced by a table lock. Two database configuration parameters have direct impact on the process of lock escalation:

- LOCKLIST - This is the number of 4 KB pages allocated in the database global memory for lock storage per node.
- MAXLOCKS - The percentage of the total lock list allowed for each application.

  Lock escalation can occur in two different situations:

  1. A single application requests a lock that will cause the application to exceed the percentage of the total lock list as defined by MAXLOCKS. The database manager will attempt to free memory space by obtaining a table lock and releasing row locks for the requesting application.
  2. An application triggers lock escalation because the total lock list is full. The database manager will attempt to free memory space by obtaining a table lock and releasing row locks for the requesting application. Note that the application being escalated may or may not have a significant number of locks. The total lock volume may be reaching a threshold because of high system activity where no individual application has reached the limit established by MAXLOCKS.

A lock escalation attempt can fail. If a failure occurs, the application that has caused the escalation attempt will receive a -912 SQLCODE. The application should be coded to handle the failure.

4.9.6 Deadlock

Deadlock occurs when two applications are waiting on a resource that the other one holds. The applications cannot complete their units of work due to conflicting lock requirements that cannot be resolved until the one of the units of work is complete.

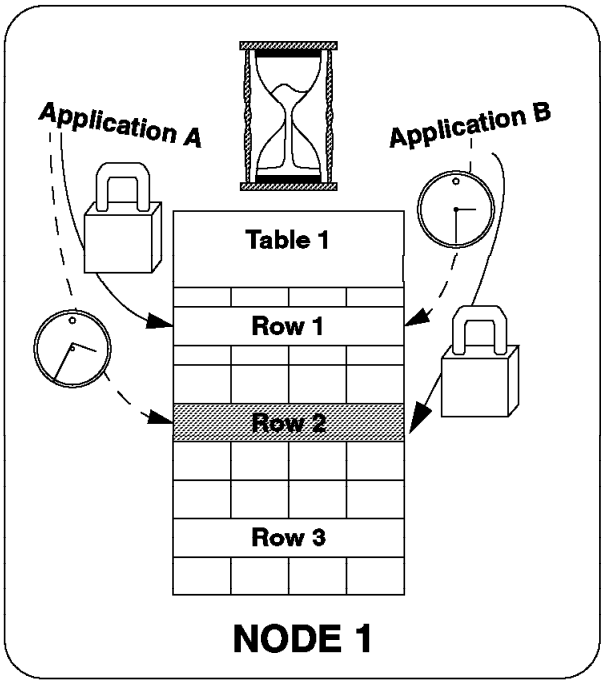


Figure 55. Deadlock on a Node

Figure 55 illustrates the concept of deadlocks in DB2 PE at the node level. Both Application A and Application B need to access Row 1 and Row 2 of Table 1. To explain further:

1. Application A obtains an X lock on Row 1 of Table 1.
2. Application B on the same node obtains an X lock on Row 2 of Table 1.
3. Application A wants an X lock on Row 2, but cannot obtain it until Application B commits.
4. Application B wants an X lock on Row 1 but cannot obtain it until Application A commits.

Neither Application A nor Application B can proceed.

This type of deadlock, caused by accessing objects in reverse order, can be reduced by establishing rules of access at an installation for highly used or highly accessed objects. In Figure 55, if both applications access Row 1 first, followed by Row 2, the first application to process would obtain an X lock on Row 1 and prevent the other from continuing at that point. The application possessing the X lock on Row 1 could proceed to get the X lock on Row 2 and complete the unit of work.

Deadlocks are handled by a background process. DB2 PE at the node level does not check for deadlocks. When waiting for a resource, the local deadlock detector will fill in a wait-for graph. The wait-for graph is described in 4.9.7, “Distributed Deadlock Detection” on page 137.

4.9.7 Distributed Deadlock Detection

The deadlock detector daemon that is provided in DB2/6000 Version 1 is not capable of detecting deadlock in a parallel environment. Figure 57 on page 138 shows a case where two transactions on different nodes, Node 1 and Node 2 are waiting for the same resources. Suppose there are two transactions, T1 and T2, each of which has two subsections, S1 and S2 as shown in Figure 56.

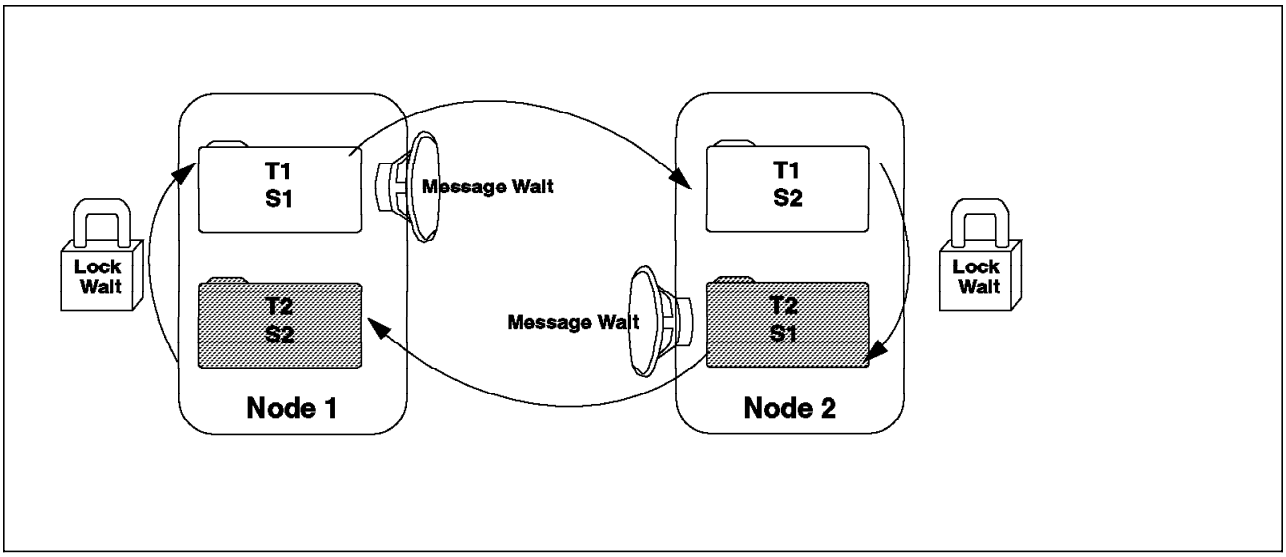
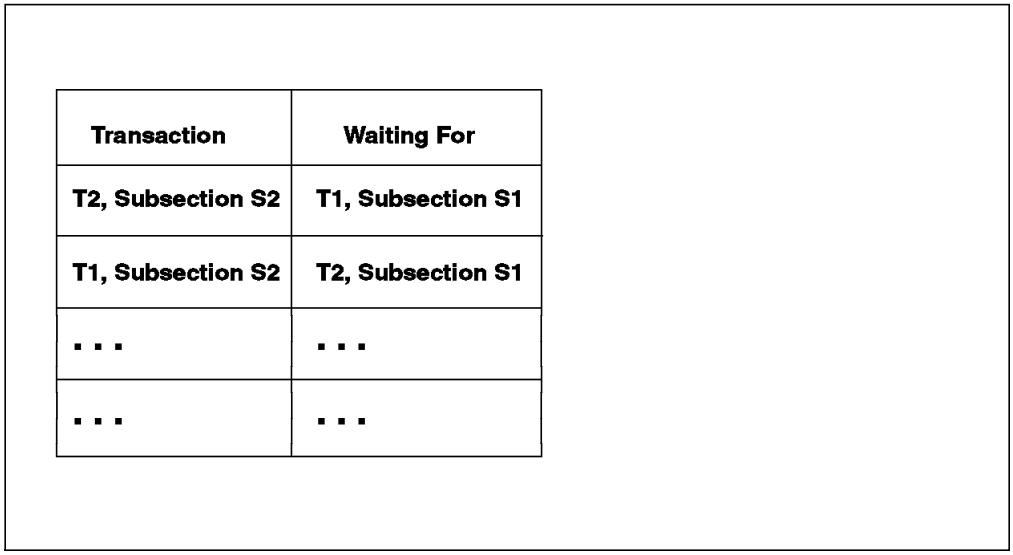


Figure 56. Distributed Global Deadlock

1. Transaction 1 updated row 1 of table 1 on node 1, and is trying to update row 10 of table 1 on node. This results in a lock wait state.
2. Transaction 2 updated row 10 of table 1 on node 2 and is trying to update row 1 of table 1 on node 1. This also results in a lock wait state.
3. No deadlock loop can be detected locally on either node.

To resolve this situation, a combined wait-for graph containing information on all the nodes is needed. Each local deadlock detector fills in information about the wait status on the respective node. This information is merged into a wait-for graph on the catalog node. The distributed deadlock detector that exists on the catalog node builds this merged graph. Figure 57 on page 138 illustrates a wait-for graph that could be built for Figure 56.



Transaction	Waiting For
T2, Subsection S2	T1, Subsection S1
T1, Subsection S2	T2, Subsection S1
...	...
...	...

Figure 57. Wait-For Graph

The wait-for graph contains a row containing what transaction is waiting for what resource. The local deadlock detector waits for the second interval (DLCHKTIME) to complete before sending the local graph to the distributed deadlock detector. This reduces the chances of a phantom deadlock. A waiting process only becomes a candidate for termination if a deadlock cycle is detected in the same lock state at the next iteration of the detectors. A victim is selected arbitrarily. The victim is automatically rolled back and returned a -911 SQLCODE. Rolling back the victim releases locks and allows the other process to continue.

4.9.7.1 Causes of Deadlocks

Although some deadlocks will occur in systems that provide both data integrity and the capability to change multiple items within a unit of work, there are certain conditions that are more prone to causing deadlocks.

- Repeatable Read
- Lock Escalation
- Lock Conversion
- Index Key Maintenance
- Catalog Modification

4.9.7.2 Lock Wait

Applications that request a lock that is not compatible with existing locks on the object will be queued for service. The database manager will not suspend the waiting application, unless a deadlock is the cause of the wait. This architecture emphasizes the importance of properly coding applications to issue commit or rollback statements within reasonable amounts of time. Because large applications that require massive amounts of data or require extensive time to complete a unit of work will be found in the parallel database environment, special concern should be shown to isolation levels and locking configuration parameters.

4.9.8 Locking Configuration Parameters

There are three parameters that directly influence your locking strategy:

- MAXLOCKS
- LOCKLIST
- DLCHKTIME

4.9.8.1 Number of Locks Per Node

The default locking method in a parallel database is row level locking. Each node in a nodegroup does the locking for the rows at that node. The database configuration parameter, MAXLOCKS, specifies the number of locks allowed per application. When this limit is reached, lock escalation occurs and locking is done at the table level. Lock escalation occurs on the table partition on a node. The default value for MAXLOCKS is 10% of the LOCKLIST. The range for MAXLOCKS is between 1 - 100. Lock escalation may affect performance, as applications wait for the table lock to be released.

When determining the size of MAXLOCKS, consider the size of the lock list as determined in the parameter, LOCKLIST. The following may be used as an example to help determine the size of MAXLOCKS:

```
MAXLOCKS = 100 *
           (512 locks per application * 44 bytes per lock * 2)
           / (LOCKLIST * 4096 bytes)
```

The above example allows any application to hold twice the average number of locks. You can increase MAXLOCKS if there are few applications run concurrently since the contention for the lock list space will be reduced. The MAXLOCKS parameter must be updated on all of the nodes in a nodegroup.

4.9.8.2 Storage for Lock Lists

The amount of storage allocated in 4 KB pages is determined by the database configuration parameter LOCKLIST. The default value is 64 4 KB pages with a range from 4 - 10,000. The lock list contains the locks held by all the applications that are currently connected to the database. This includes both row locks and table locks.

Each lock needs 44 bytes of space in the lock list. The size of LOCKLIST may be estimated by the following formula:

```
LOCKLIST =  (average number of locks per application
            * 44 * MAXAPPLS)
            / 4096
```

Note: In the formula, 512 may be substituted as an average number of locks per application.

The database configuration parameter, MAXAPPLS, is the number of applications that can concurrently connect to the database. This provides control over the amount of private memory used for

database applications. This parameter limits the number of active applications against the database on a node, whether that node is the coordinator node for the application or not.

When the percentage of the lock list used by one application reaches MAXLOCKS, the database manager does a lock escalation. The escalation process does not require much time, but table locking may affect concurrency. This can reduce overall database performance when subsequent access is attempted against the locked tables. To reduce the number of locks held, you can:

- Do frequent commits to release locks.
- Lock the entire table when performing updates via the `lock table` SQL statement. This only requires one lock and prevents other users from interfering with the updates. However, concurrency to the data will be reduced.
- Use Cursor Stability or Uncommitted Read as the isolation level when possible to decrease the number of locks obtained.

When the lock list is full, performance may decrease since lock escalation produces more table locks and fewer row locks. This can reduce concurrency on shared objects in the database. Also, there can be more deadlocks between applications because they are all waiting on a small number of table locks. This can cause transactions being rolled back. An application will receive a -912 SQLCODE when the maximum number of lock requests has been reached for the database. Coding your application to handle the error situation is recommended.

4.9.8.3 Checking for Deadlocks

The database configuration parameter, DLCHKTIME, tells the database manager how often to check for deadlocks among the applications that are connected to a database. This parameter applies only to the catalog node. However, make sure that all the nodes in the nodegroup have the same value for DLCHKTIME. If a smaller value is found on the catalog node than on some of the other nodes, phantom deadlocks may occur. If the catalog node has a larger value than the other nodes, it may appear that more than two intervals pass before a deadlock is detected.

The default value for DLCHKTIME is 10 seconds. If you increase this parameter, you decrease the frequency of deadlock detection. If your application is read only, this may be sufficient. If your application changes data, an increase in DLCHKTIME may result in an increase in the time that an application program must wait for deadlock to be resolved.

If you decrease DLCHKTIME, you increase the frequency of deadlock detection. This can result in a decrease in the amount of time an application waits for deadlock resolution. It does increase the amount of time the database manager spends checking for deadlocks. If this interval is too small, performance can be affected because the database manager is consuming resources to check for deadlock. If you increase DLCHKTIME, you may have to increase MAXLOCKS and LOCKLIST to avoid unnecessary lock escalation.

Chapter 5. Parallel Utilities

DB2 Parallel Edition provides a number of tools and utilities to assist a database administrator in managing and monitoring databases The following will be discussed in this chapter:

- Executing Commands on Multiple Nodes
- Segment Manager Tool
- Data Splitting and Loading
- Autoloader Utility
- Import/Export Utility
- Adding Nodes
- Dropping Nodes
- Data Redistribution
- Runstats Utility
- Reorgchk Utility
- Reorganization
- Backup and Restore
- Recovery
- Governor Utility
- db2batch Tool
- Database Director Utility

5.1 Executing Commands on Multiple Nodes

DB2 Parallel Edition provides a utility so that commands can be issued at one node to execute on all or on multiple nodes. The utility is invoked by one of the following:

- The `rah` command
- The `db2_all` command.

While the `db2_all`, and `rah` commands both use the same executable, the utilities differ in the following ways:

1. The `db2_all` command causes a command to execute at all physical and logical nodes specified in the $HOME/sqllib/db2nodes.cfg file.
2. The `rah` causes a command to execute at all physical hosts.

For more information about these commands:

- Type `rah "?"` at the command line and press **ENTER**.
- See the README file in the $HOME/sqllib/misc directory. The file name is rahREADME.

The following example illustrates the difference between using rah and db2_all. Here, the $HOME/sqllib/db2nodes.cfg file consists of four logical nodes on three physical hosts.

```
0  HOST0  0
1  HOST1  0
2  HOST2  0
3  HOST2  1
```

We will execute the date command in our parallel environment using rah:

```
rah "date"
```

The following output is returned:

```
rah:  HOST0
Wed Nov 15 21:48:33 EST 1995
HOST0: date completed ok

rah:  HOST1
Wed Nov 15 21:48:33 EST 1995
HOST1: date completed ok

rah:  HOST2
Wed Nov 15 21:48:33 EST 1995
HOST2: date completed ok
```

Next, we will issue the same date command using db2_all:

```
db2_all "date"
```

The following output is returned:

```
rah:  HOST0
rah:  HOST1
rah:  HOST2
rah:  HOST2

HOST1: Wed Nov 15 21:51:51 EST 1995
HOST1: date completed ok

HOST2: Wed Nov 15 21:51:52 EST 1995
HOST2: date completed ok

HOST2: Wed Nov 15 21:51:57 EST 1995
HOST2: date completed ok

HOST0: Wed Nov 15 21:51:50 EST 1995
HOST0: date completed ok
```

Note: Both rah and db2_all require double quotes surrounding the command(s) to be executed.

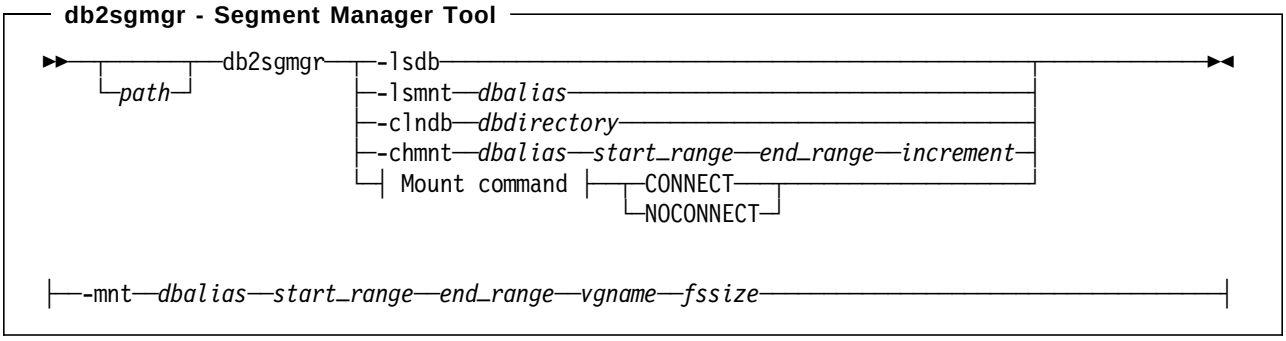
5.2 Segment Manager Tool

The Segment Manager Tool is a front-end utility that DB2 PE system administrators can use to extend the database size beyond the operating system limits imposed by AIX. This tool can only be run against local databases which are cataloged in the system database directory. The Segment Manager Tool can only be executed from server nodes.

This tool and all of its options may be accessed from the AIX command line by entering db2sgmgr -? You must have SYSADM authority to execute the Segment Manager Tool.

This tool is also accessible through the SMIT interface by selecting the Applications entry on the SMIT main menu. However, accessing the Segment Manager Tool through the SMIT utility must be done on all nodes individually in DB2 PE. This section shows the Segment Manager Tool issued from the command line for all nodes.

The full syntax for db2sgmgr is as follows:



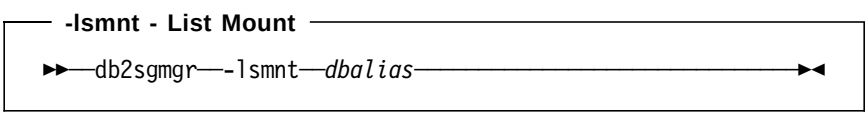
We will follow through an example of allocating the segment directories of an existing database onto separate file systems. For this example, a database has been created with the following command:

```
db2 create database asample on /db2 NUMSEGS 10 SEGPAGES 64
```

This database will be loaded with 800 MB of data on each node. There are 8 nodes defined in the nodegroup. There will be 10 segment directories created on each node defined in the nodegroup of the database. The segment size will be set to 64 4 KB pages. After the data has been loaded, we will check the initial allocation of segment directories to file systems for our given database (in this case, asample).

5.2.1 Segment Directory File System Information

This option provides the mount status information for each segment directory of a particular database. The syntax for the command to display the mount status information for each segment directory of a database is as follows:



For our example (the database is asample), the command is:

```
db2_all "db2sgmgr -lsmnt asample"
```

```
rah: host0

Database: ASAMPLE
Path    : /db2/db2puser/NODE00000/SQL00002

Segment Directory: 0
File System Mount Point: /db2
Size (KB)             : 819200
Free (KB)             : 14720
Percent used          : 98

Segment Directory: 1
File System Mount Point: /db2
Size (KB)             : 819200
Free (KB)             : 14720
Percent used          : 98

[ Segments 2-8 omitted - they are identical in detail ]
[ Output for node 0 - the other nodes are identical in detail ]

Segment Directory: 9
File System Mount Point: /db2
Size (KB)             : 819200
Free (KB)             : 14720
Percent used          : 98

SEG0000I  The command completed successfully.
```

Figure 58. Display Segment Directory File System Information

This shows:

- This database is split over 10 segment directories (0 to 9).
- All the segment directories are mounted on the same filesystem /db2.
- This one filesystem (/db2) is 800 MB in size and is 98 percent used.

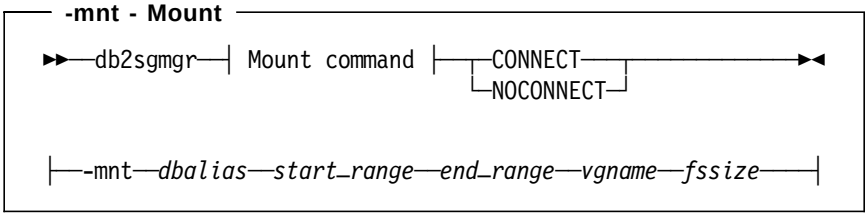
This would be a good time to move these individual segment directories into different file systems.

5.2.2 Mounting Segment Directory to File System

The mount option allocates and mounts Journaled File Systems (JFS) over database segment directories.

You can use this command to mount all ten segment directories onto 10 file systems on all 8 nodes in the nodegroup, one segment directory per file system. The syntax is as follows:

-mnt - Mount



For our example, the command would be:

```
db2_all "db2sgmgr -mnt asample 0 9 db2vg 204800 CONNECT"
```

Note that the segment directory range starts from 0, so the 10 segment directories are referenced as 0 to 9.

The size of the file systems to be created is set to 204800 4 KB pages, which is 800 MB. It would be possible to have many differently sized file systems simply by specifying each segment directory individually rather than entering a range.

For this command to work successfully, there must be 800 MB of free space in the db2vg volume group on each node.

The CONNECT option of the command will perform an exclusive connect to the database. This means that no other connection is allowed to the database for the duration of this command. Normally, you would set this to NOCONNECT when a connection to the database is not possible, for instance, a corrupt database. If you want the db2sgmgr with mount option to be executed on all nodes in parallel, set the option to NOCONNECT. You must, however, ensure that there are no other connections to the database on those nodes until the command has completed.

The output of the command is similar to the following:

```
rah: host0
SEGO019I  Allocating a file system for database segment directory
/db2/db2puser/NODE00000/SQL00002/SQLS0000.

SEGO019I  Allocating a file system for database segment directory
/db2/db2puser/NODE00000/SQL00002/SQLS0001.

[ Segments 2-8 omitted - they are identical in detail ]
[ Only node 0 is displayed ]

SEGO019I  Allocating a file system for database segment directory
/db2/db2puser/NODE00000/SQL00002/SQLS0009.

SEGO000I  The command completed successfully.
```

Figure 59. Mount Segment Directory File Systems

To display the segment directory file system information again, enter:

```
db2_all "db2sgmgr -lsmnt asample"
```

The output is as follows:

```
rah: host0

Database: ASAMPLE
Path    : /db2/db2puser/NODE00000/SQL00002

Segment Directory: 0
File System Mount Point: /db2/db2puser/NODE00000/SQL00002/SQLS0000
Device                : /dev/lv02
Size (KB)             : 819200
Free (KB)             : 744200
Percent used          : 9

Segment Directory: 1
File System Mount Point: /db2/db2puser/NODE00000/SQL00002/SQLS0001
Device                : /dev/lv03
Size (KB)             : 819200
Free (KB)             : 734200
Percent used          : 10

[ Segments 2-8 omitted - they are similar in detail ]
[ Only output for node 0 is shown. ]

Segment Directory: 9
File System Mount Point: /db2/db2puser/NODE00000/SQL00002/SQLS0009
Device                : /dev/lv11
Size (KB)             : 819200
Free (KB)             : 744100
Percent used          : 9

SEG0000I  The command completed successfully.
```

Figure 60. Display Segment Directory File System Information

Each segment directory (SQLS0000 to SQLS0009) is now mounted on its own file system, each with 800 MB of allocated space. So now, each file system is approximately 9 to 10 percent utilized, compared with 98 percent utilization before.

This database is now set up to accommodate a data volume of up to 800 MB per node. The exact capacity depends on how evenly the data files are spread across the segment directories, but it will be a figure slightly less than 800 MB.

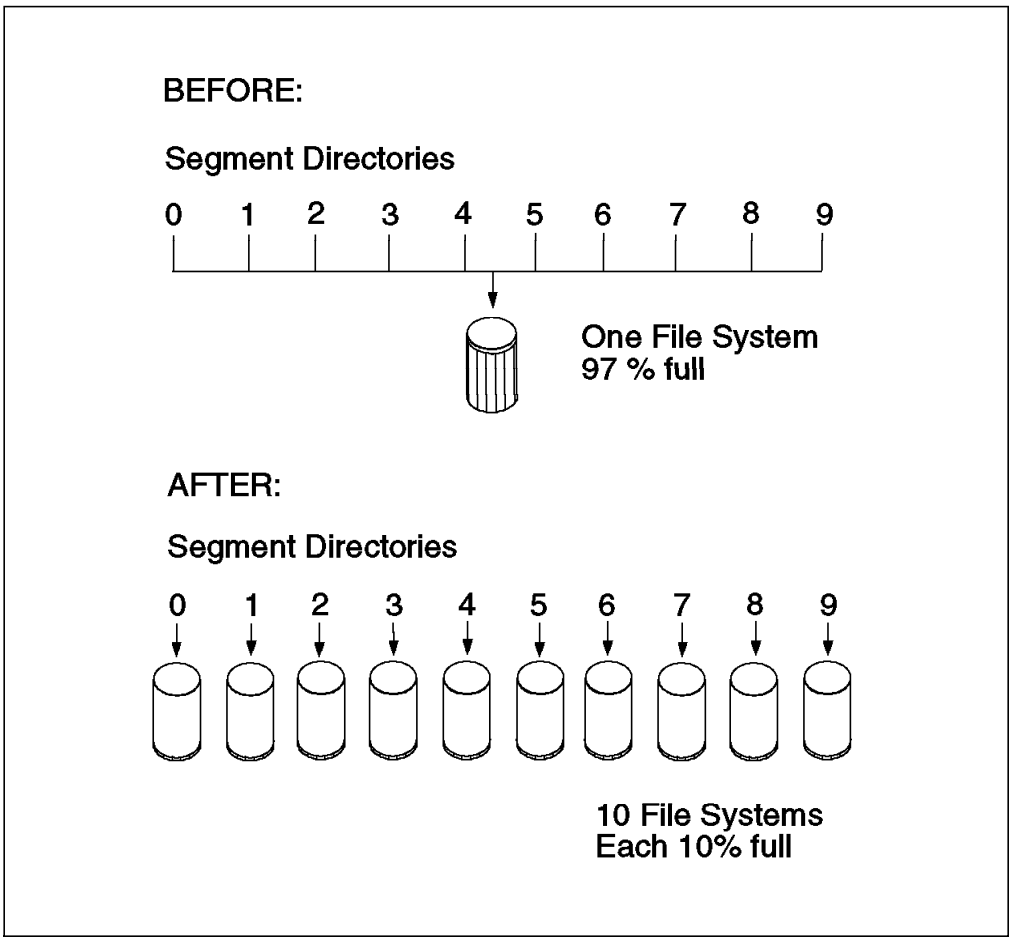


Figure 61. Segmented Directories Example

5.2.3 Increase Segment Directory File System Size

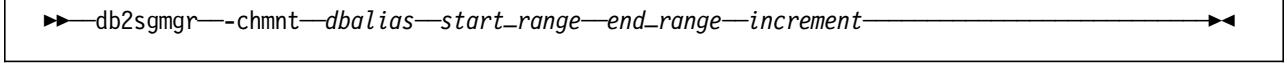
If one of the file systems becomes close to filling up, you will need to increase the size of that file system.

It is important to know that a file system can be easily extended (increased) but to decrease its size is a much longer procedure which entails:

- Copying the data to another location
- Deleting the original file system
- Creating a new smaller file system
- Copying the original data to this new file system

To increase the size of a segment directory file system, the change mount option will be used. The change mount option increases the size of mounted database segment file systems by an increment specified in blocks of 4 KB for the database. The range of the file systems to be increased is specified by the `start_range` and the `stop_range` values. The complete syntax is as follows:

-chmnt - Change Mount



In our example, we will increase the segment directory file system for SQLS0003. The command is as follows:

```
db2_all "db2sgmgr -chmnt asample 3 3 25600"
```

In this example, the file system for segment directory SQLS0003 will be increased by 25600 4 KB pages, or 100 MB. The output of the command is as follows:

```
rah: host0

SEG0020I  Increasing the File System Size for the database segment
directory /db2/db2puser/NODE00000/SQL00002/SQLS0003.
Filesystem size changed to 921600

[ Only output for node 0 is shown. ]

SEG0000I  The command completed successfully.
```

Figure 62. Increase Segment Directory File System Size

Now the file system is 921600 KB or 900 MB.

5.2.4 Cleanup Database Directory After Dropping Database

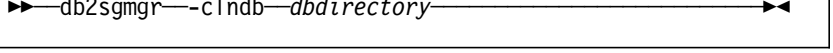
This command deallocates the segment directories that were mounted on different file systems and removes the database directory (after the database has been dropped). This command is necessary to clean up the segment file systems after dropping a database.

When the database is dropped, the following message is generated:

```
db2 => drop database asample
SQL1137W  The database manager was unable to remove the database path when
dropping database "asample". Cleanup is required.
```

This deletes all the data files (.DAT, .IDX, .LF) for the database, but leaves the file systems intact. The complete syntax to clean up the database directory is as follows:

-clndb - Cleanup Database Directory



The parameter, dbdirectory, specifies the full database directory path. Cleanup will not proceed unless the directory specified

contains no files other than the database segment subdirectories. The following example will clean up the directory on NODE1:

```
db2sgmgr -clndb /db2/db2puser/NODE00001/SQL00002
```

The output for node 1 is as follows:

```
Before command completion, additional instructions may appear below.

Database directory path: /db2/db2puser/NODE00001/SQL00002

SEG0017I  Deallocating /db2/db2puser/node00001/SQL00002/SQLS0000.
SEG0017I  Deallocating /db2/db2puser/node00001/SQL00002/SQLS0001.
SEG0017I  Deallocating /db2/db2puser/node00001/SQL00002/SQLS0002.
SEG0017I  Deallocating /db2/db2puser/node00001/SQL00002/SQLS0003.
SEG0017I  Deallocating /db2/db2puser/node00001/SQL00002/SQLS0004.
SEG0017I  Deallocating /db2/db2puser/node00001/SQL00002/SQLS0005.
SEG0017I  Deallocating /db2/db2puser/node00001/SQL00002/SQLS0006.
SEG0017I  Deallocating /db2/db2puser/node00001/SQL00002/SQLS0007.
SEG0017I  Deallocating /db2/db2puser/node00001/SQL00002/SQLS0008.
SEG0017I  Deallocating /db2/db2puser/node00001/SQL00002/SQLS0009.

SEG0018I  Removing database directory /db2/db2puser/NODE00001/SQL00002/.
```

Figure 63. Cleanup Database Directory After Dropping a Database

So, all of the file systems allocated to the segment directories SQLS0000 to SQLS0009 are removed, and the database directory itself is deleted.

If there were any other (non-database) files in the database directory, (in our example, SQLS00002) the command would fail with the following message:

```
SEG0036N  The database directory /db2/db2puser/NODE00001/SQL00002
is not empty.
```

5.3 Data Splitting and Loading

In DB2 Parallel Edition, there are different methods to move data from and to the database. This section provides detail on the DB2 Parallel Edition utilities that allow you to populate a table without logging.

5.3.1 Populate a Table without Logging

This method will load data without logging. For tables larger than 50 MB and spread over multiple nodes, this method can provide considerable performance gains over the import utility. To populate a table without logging on a multi-node system, you can:

1. Partition the data for each node using db2split.
2. Send the output files which are split using db2split to the appropriate node by the file transfer program (FTP).
3. Load the data from the split output files to the multi-node table.

For tables in single-node nodegroups, the data does not have to be split using db2split, even if the table is defined with a partitioning key. In this case, specify the noheader option of the load utility to load the single-node table.

5.3.2 Partitioning Data with db2split

DB2 Parallel Edition provides the db2split tool to partition data for each node. This data splitting tool can be executed on MVS, VM or AIX platforms. Typically, it is executed on the platform where the source file resides. It is highly recommended that the input data be sorted based on the partitioning key columns or the most-used index columns to achieve a high cluster ratio.

The db2split program requires a configuration file. Refer to the *DB2 Parallel Edition Administration Guide* for the list of parameters you can specify in the configuration file.

To display the list of options for db2split, issue:

```
db2split -h
```

There are two ways you can use db2split:

- Produce a customized partitioning map that balances the data across nodes for the table.

  Typically, this method is used to populate a table in a new nodegroup which does not yet contain data. The steps are as follows:

  1. Execute d2split in analyze mode to create a partitioning map. It is recommended to specify the data file for the largest table that will be in the same nodegroup as the input file.
  2. Execute db2split again to partition the data using the partitioning map that was just created.
  3. Issue the `redistribute nodegroup` command to catalog the partitioning map that was just created for the nodegroup in the database.
  4. Create the table in the nodegroup with the `create table` SQL statement.

- Use the partitioning map of an existing nodegroup.

  Typically, this method is used to populate a table in an existing nodegroup. The steps are as follows:

  1. Create the table in the existing nodegroup with the `create table` SQL statement.
  2. Execute db2gpmap to obtain the partitioning map of the nodegroup from the catalog node.
  3. Execute db2split to partition the data using the partitioning map that was just obtained.

5.3.3 db2split Example

This section provides an example of a data file and the table into which data is to be loaded. The following example illustrates the different results you may get when using a customized partitioning map versus using a default partitioning map.

Assume that you have a delimited data file called mydata with 320,000 records, as follows:

```
0,0,0,"xxxxxxxxxx","xxxxxxxxxx"
1,0,0,"xxxxxxxxxx","xxxxxxxxxx"
       ......
       ......
319998,3199,0,"xxxxxxxxxx","xxxxxxxxxx"
319999,3199,0,"xxxxxxxxxx","xxxxxxxxxx"
```

Also, assume that you want the table to be distributed on 4 nodes (0,1,2,3). The following is an example of the nodegroup and table definition:

```
CREATE NODEGROUP ng4 ON NODES (0,1,2,3);

CREATE TABLE test1 (col1 INTEGER,
                    col2 INTEGER,
                    col3 INTEGER,
                    col4 VARCHAR(40),
                    col5 VARCHAR(40))
                IN ng4
                PARTITIONING KEY (col2) USING HASHING;
```

5.3.3.1 Partitioning Data Using a Customized Map

The following shows the configuration file, analyze.cfg, for db2split that will generate a customized partitioning map on an AIX system:

```
(1)  Description=test1
(2)  RecLen=200
(3)  InFile=/data1/mydata
(4)  Nodes=(0,1,2,3)
(5)  Mapfilo=/home/db2puser/split/OutMap
(6)  LogFile=/home/db2puser/split/Log.a,w
(7)  CDelimiter=,
(8)  RunType=ANALYZE
(9)  Msg_Level=WARN
(10) Check_Level=NOCHECK
(11) Partition=col2,2,,,N,INTEGER
(12) Trace=0
```

Note:

1. You can provide any text in the Description field. In this case, the table name, test1, is used.
2. Since the input data is delimited, the RecLen field is optional. If this field is not specified, db2split reserves 32700 bytes for each record, which can impact performance.
3. In this example, the nodegroup for the test1 table is defined on nodes 0-3; the Nodes field must be consistent with the table nodegroup.

4. The Mapfilo field indicates where the output partitioning map will be stored.
5. The LogFile field indicates where the output messages from db2split will be written.
6. The CDelimiter field indicates the data in the input file is delimited with a comma (,).
7. The RunType field indicates db2split will analyze the data using the node information from the Nodes field to produce a customized partitioning map that achieves an even data distribution on the nodes.
8. The Msg_Level field indicates db2split stops at every warning message.
9. The Check_Level field indicates db2split does not check for the truncation of records at I/O.
10. The Partition field indicates the information of the partitioning key. The detail is as follows:
    - The col2 is the column name of the table used as the partitioning key.
    - The 2 is the cardinal value of the partitioning field in the record of the input file.
    - The N indicates the data is nullable.
    - The last fields in the argument indicate the data type of the column in the partitioning key. In this case, the data is converted to a 4-byte integer for hashing into the partition index.

To execute db2split with the configuration file, analyze.cfg, issue the following:

```
db2split -c analyze.cfg
```

Note: This will generate the customized partitioning map. The map will be written to the directory /home/db2puser/split with the filename of OutMap.

The following shows the configuration file, sample1.cfg, using the customized partitioning map, /home/db2puser/split/OutMap, generated from the previous steps to partition the data:

```
(1)  Description=test1
(2)  RecLen=200
(3)  InFile=/data1/mydata
(4)  Mapfili=/home/db2puser/split/OutMap
(5)  LogFile=/home/db2puser/split/Log.sample1,w
(6)  OutFile=/data1/output
(7)  CDelimiter=,
(8)  RunType=PARTITION
(9)  Msg_Level=WARN
(10) Check_Level=NOCHECK
(11) Partition=col2,2,,,N,INTEGER
(12) Trace=1
```

Note:

- The Mapfili field in line 4 indicates the input partitioning map is /home/db2puser/split/OutMap.
- The OutFile field in line 6 indicates db2split will write the results of the split data for nodes 0-3 to the /data1 directory. The output file name is prefixed with the word “output” and suffixed with 00000-00003. In this example, the output files are:
  - output.00000
  - output.00001
  - output.00002
  - output.00003
- The RunType field indicates db2split will partition the data.

To execute db2split with the configuration file sample1.cfg:

```
db2split -c sample1.cfg
```

This produces the following messages in the log file, /home/db2puser/split/Log.sample1:

```
test1 >Log file opened successfully
test1 > Start time: Fri Oct  6 15:31:50 1995
test1 > Input file /data1/mydata opened successfully
test1 > Input maximum record length :200
test1 > Program is running with NOCHECK level
test1 > The string delimiter is :<">
test1 > Tracing 0 delimited (delimiter <,>) record(s)
test1 > Input map file /home/db2puser/split/OutMap opened successfully
test1 > for reading
test1 > Getting partitioning map...done
test1 > The Run Type is PARTITION
test1 > Output partitioning map file not used
test1 > The message level is WARN
test1 > Distribution file name: DISTFILE
test1 > Distribution file DISTFILE opened successfully for writing
test1 > Working on  1 keys.
test1 > COL2    Start:   0  Len:   0  Position:   2  Type: N INTEGER 1
test1 > Output files will be /data1/output.00xxx
test1 > All output files opened successfully
test1 > Processed        50000
test1 > Processed       100000
test1 > Processed       150000
test1 > Processed       200000
test1 > Processed       250000
test1 > Processed       300000
test1 > Writing distribution map to DISTFILE
test1 > Total record count:    320000        2
test1 > Total record discarded:         0       3
test1 > Stop time: Fri Oct  6 15:32:36 1995
test1 > Elapsed time:  0 hours,  0 minutes, 46 seconds
test1 > Throughput: 6956 records/sec
test1 > Record counts for output nodes:  4
Node: 0: Record count: 80000
Node: 2: Record count: 80000
Node: 1: Record count: 80000
Node: 3: Record count: 80000
test1 > Complete.
Program ran successfully with 0 warning message(s) and 0
discarding record(s)
```

1 Indicates the information of the partitioning column.

2 Indicates 320,000 records were processed.

3 Indicates 0 records were discarded.

4 Indicates the total record counts for each output nodes. You can see the data is split evenly among 4 output files. Each of the output files contains 80,000 records.

5.3.3.2 Partitioning Data Using Nodes

The db2split tool uses the node number specified in the Nodes field in the configuration file to generate a partitioning map and partitions the data based on that partitioning map. The following shows the configuration file, sample2.cfg, that is used to partition data on AIX:

```
(1)  Description=test1
(2)  RecLen=200
(3)  InFile=/data1/mydata
(4)  Nodes=(0,1,2,3)
(5)  LogFile=/home/db2puser/split/Log.sample2,w
(6)  OutFile=/data1/output2
(7)  CDelimiter=,
(8)  RunType=PARTITION
(9)  Msg_Level=WARN
(10) Check_Level=NOCHECK
(11) Partition=col2,2,,,N,INTEGER
(12) Trace=0
```

To execute db2split with the configuration file, sample2.cfg, issue the command:

```
db2split -c sample2.cfg
```

This produces the following messages in the log file, /home/db2puser/split/Log.sample2:

```
test1 >Log file opened successfully
test1 > Start time: Fri Oct  6 16:12:05 1995
test1 > Input file /data1/mydata opened successfully
test1 > Input maximum record length :200
test1 > Program is running with NOCHECK level
test1 > The string delimiter is :<">
test1 > Tracing 0 delimited (delimiter <,>) record(s)
test1 > Getting partitioning map...done
test1 > The Run Type is PARTITION
test1 > Output partitioning map file not used
test1 > The message level is WARN
test1 > Distribution file name: DISTFILE
test1 > Distribution file DISTFILE opened successfully for writing
test1 > Working on  1 keys.
test1 > COL2    Start:   0  Len:   0  Position:   2  Type: N INTEGER  1
test1 > Output files will be /data1/output2.00xxx
test1 > All output files opened successfully
test1 > Processed        50000
test1 > Processed       100000
test1 > Processed       150000
test1 > Processed       200000
test1 > Processed       250000
test1 > Processed       300000
test1 > Writing distribution map to DISTFILE
test1 > Total record count:    320000     2
test1 > Total record discarded:         0     3
test1 > Stop time: Fri Oct  6 16:12:53 1995
test1 > Elapsed time:  0 hours,  0 minutes, 48 seconds
test1 > Throughput: 6666 records/sec
test1 > Record counts for output nodes:  4
Node: 3: Record count: 79800
Node: 2: Record count: 80000
Node: 1: Record count: 77400
Node: 0: Record count: 82800
test1 > Complete.
Program ran successfully with 0 warning message(s) and 0 discarding record(s)
```

1 Indicates the information of the partitioning column.

2 Indicates 320,000 records were processed.

3 Indicates 0 records were discarded.

4 Indicates the total record counts for each node's output. You can see the data is not split as evenly as it was in the example using the customized partitioning map.

5.3.3.3 Header Information from the Output File

The data partitioning program, db2split, writes header information to each data output file that it creates. The header information contains:

1. The node number - this is found in the first line of the file.
2. Beginning at the second line of the file is the block of 4096 integers of the partitioning map in free format.
3. The ===PDB=== line indicates the end of the block entries of the partitioning map.
4. The fourth line has the entry for the number of partitioning columns.
5. The fifth line is the block of partitioning columns that forms the partitioning key. Each line indicates:
   a. The name of the column

b. The data type

c. The data length

d. The position of the column

Note: The header information for each file will be used by the load utility to ensure that data goes to the correct location.

The following shows the header information of the output files, /data1/output.00000, partitioned by using the customized map described in 5.3.3.1, “Partitioning Data Using a Customized Map” on page 151:

```
0     1
3 3 1 2 0 3 3 3 2 0 1 1 3 1 0 3 3 3 0 0 ......  2
........
..............3 0 1 2 3 0 1 2 3
===PDB===             3
1                                4
Key: COL2 497  4  2  0                          5
===PDB===             6
0,0,0,"xxxxxxxxxx","xxxxxxxxxx"                 7
   ........
   ........
319899,3198,0,"xxxxxxxxxx","xxxxxxxxxx"
```

1 Shows the node number of 0. It indicates that this file should be loaded to the table partition on node 0.

2 Shows the block of 4096 entries for the partitioning map. You can see the node numbers in the map are (0,1,2,3).

3 Indicates the ending of the block of partitioning map entries and the beginning of the header for the partitioning columns.

4 Shows the partitioning key has only one column.

5 Shows the following:

- The COL2 indicates the name of the partitioning column.
- The 497 indicates the data type is integer with NULL.
- The 4 indicates the data length of an integer field.
- The 2 indicates COL2 is the second item in the file.

6 Shows the end of the block of entries for the partitioning column.

7 This is the beginning of the data to be loaded.

5.3.4 Sending Partition Files to Appropriate Nodes

After you create the partition files, use FTP to send each individual file to the appropriate node. The following shows the steps to send the output files, /data1/output.00001, to the host with the hostname of host1 on AIX:

1. To connect to the host where the file is to be sent, issue:

   ```
   ftp host1
   ```

2. Supply the username and password.
3. Change directory to the one where the file will be stored by issuing:

   ```
   cd /data1
   ```

4. Store the file to the directory, /data1, by issuing:

   ```
   put output.00001
   ```

5. To disconnect from the host, issue:

   ```
   quit
   ```

Note: FTP does EBCDIC to ASCII translation if needed.

5.3.5 Load Utility

The Load utility is used to load data into a table from a file without logging. You must be connected to the node that has the database partition into which you want to load the data.

For tables defined in single-node nodegroups, the input file does not have to be created and split using db2split.

For tables defined in multi-node nodegroups, the input file for the load must be created and split using db2split. You must have SYSADM or DBADM authority to use the Load utility.

5.3.5.1 Considerations for the Load Utility

There are a number of steps to consider before beginning the load operation and after completion of the load operation. Before beginning the load operation, it is recommended that the following steps be performed:

1. Know the format of your data in the input files for the load utility. The format must be either ASC or DEL. (IXF and WSF are not supported)
2. Sort the input data before splitting, if necessary.
3. For tables in single-node nodegroups, it is not required to split the input data using db2split.
4. For tables in multi-node nodegroups, you must split the input data using db2split and ensure the following:
   - The node number in the header of the split file is the same as the node number where the file is being loaded.
   - The partitioning map in the header of the split file is the same as the partitioning map used by the table being loaded.
   - The name of the column, and the type and length of the partitioning key specified in the header of the split file is the same as the definition of the table being loaded.
5. If you are loading a table that already contains data, you should have a backup copy of your database.
6. If the table currently has indices defined (primary key or otherwise), you must first drop them.

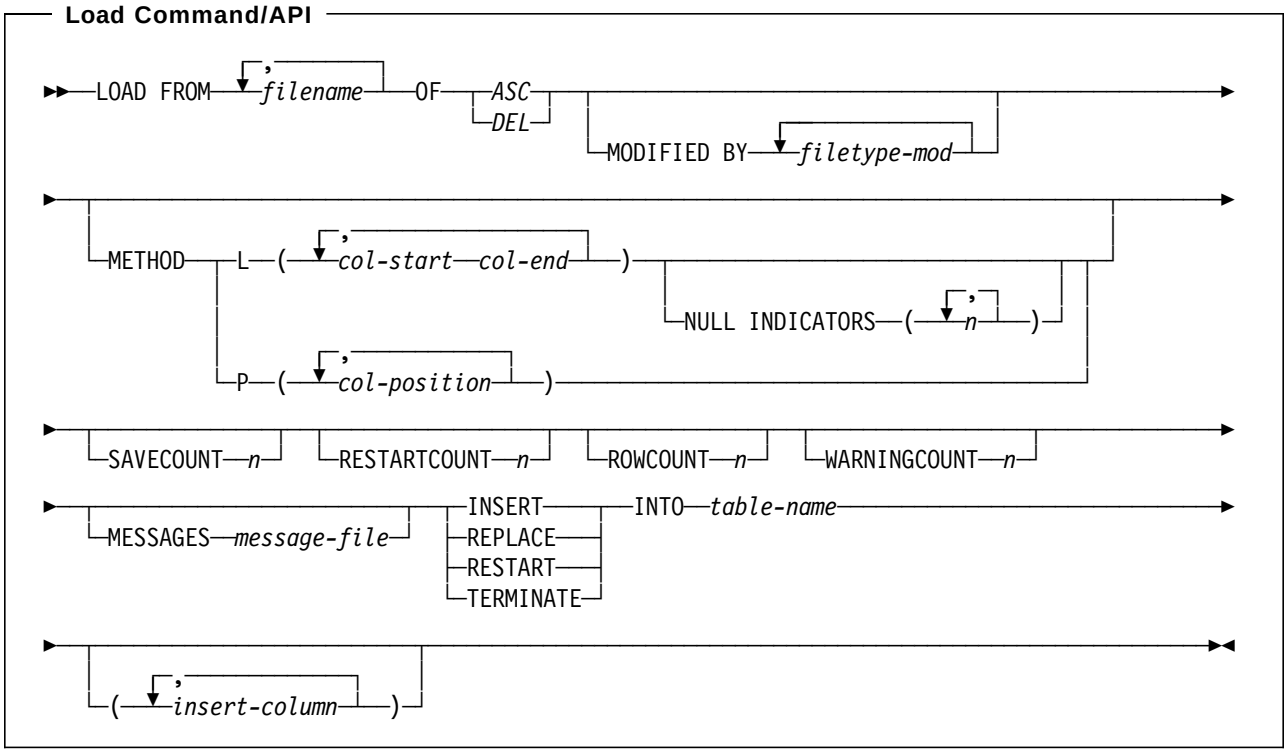
7. If you are loading into a new table, make sure the target table is created.
8. Place the load command in a file that can be edited and then executed from the command line.

After completion of the load operation, it is recommended the following steps be performed:

1. Check the message file from the load to make sure there are no errors.
2. Count the records in the table by issuing the query `select count(*)` against the table.
3. If all the load operations have completed and the row-count of the tables is correct, back up the database before using.

5.3.5.2 Syntax of the Load Command

The full syntax of the load command is:



For a description of all of the parameters, see the *DB2 Parallel Edition for AIX Administration Guide and Reference*. The following parameters are described in detail:

filename This parameter identifies the source of the data being loaded. The file or device must be on the same node as the table being loaded. If several data sources are identified, they will be loaded in sequence.

ASC

Non-delimited ASCII requires the data to be aligned in columns. An example of an ASC format is:

```
Smith, John       1235       400.50
Clark, Bob        2356       500.50
Williams, Jones   351        5000.50
```

DEL

Delimited ASCII data values are separated by a special delimited character. This character must only be used for this purpose within the file. An example of a DEL format is:

```
'Smith, John';1235;400.50
'Clark, Bob';2356;500.50
'Williams, Jones';351;5000.50
```

where the single straight quote mark (') is a character string delimiter; the semicolon (;) is a column delimiter, and the dot (.) is a decimal point.

MODIFIED BY

Specifies additional information unique to the DEL and ASC file format specified in the *filetype* parameter.

- For the ASC file format:
  - A T in the *filetype-mod* string indicates that trailing blanks spaces are truncated. If the *filetype-mod* parameter is not specified in the command string, trailing blank spaces are kept.
  - You can specify the RECLEN=xxx option as the *filetype-mod*. The xxx integer can be no larger than 32,767. Instead of using the new-line character to indicate the end of a row, xxx characters are read in for each row.
  - If you specify NULIND, or NULLIND, the character that immediately follows is the null indicator. You cannot specify an equal sign (=), blank space ( ), or comma (,) as the null indicator.
  - You can specify the DUMPFILE=*fully-qualified-filename* option, and any rejected row will be written to this file. The row that is written to the file will not exactly match the row in the source data.
- For the DEL (delimited ASCII) file format, *filetype-mod* selects characters to override the following options:
  - Column delimiters, which are commas (,) by default. Specifying COLDEL followed by one character causes the

specified character to be used to signal the end of a column.

- Character string delimiters, which are double quotation marks (″) by default. Specifying CHARDEL followed by one character causes the specified character to be used to enclose a character string.
- Decimal point characters, which are periods (.) by default. Specifying DECPT followed by one character causes the specified character to be used as a decimal point.

METHOD L If the source data is an ASC file, use the L parameter to identify the first and last byte of each column of data to be loaded.

METHOD P If the source data is a DEL file, use the P parameter to identify the numbered order of the columns to be loaded.

SAVECOUNT n This parameter is used to establish consistency points during a load after every n rows. A message is issued for each consistency point. The consistency messages are written to the message file. If the value for n is small, there will be processing overhead and I/O will be impacted. The default value is 0, which means no consistency points are established. When you specify SAVECOUNT, ensure that the minimum number of pages represented by n is 500.

RESTARTCOUNT n Skips n rows and restarts the load at n+1.

ROWCOUNT n Loads n rows.

WARNINGCOUNT n Stops the load after n warnings.

message-file Specifies the location for warning and error messages that occur during the load. If the message file is omitted, the messages are written to standard output. If the complete path to the file is not specified, load uses the current directory as the destination. If the name of a file that already exists is specified, load appends the information to it.

INSERT Adds the loaded data to the table without changing the existing table data.

REPLACE Deletes all existing data in the table and inserts the loaded data.

RESTART This parameter is used to restart the load operation if an error occurs. The load continues from the last valid consistency check that the utility could establish. Restart is designed to be used if the message file

contains a pair of consistency check messages. If you find the pair of messages, you can issue this parameter together with the RESTARTCOUNT parameter to continue loading the table.

TERMINATE This parameter is used to clean up data if an error occurs. It is designed to be used if the message file does not contain a pair of consistency check messages. After you run load terminate, check the message file for the last pair of consistency check messages. When you determine how many rows were successfully loaded from the message file, you can continue the load with either a load RESTARTCOUNT restart operation or a load insert operation.

table-name Specifies the target table within the database in which the data is to be loaded. The table cannot be a system table.

insert-column Specifies the name of a column in the table into which the data is to be inserted.

5.3.6 Examples Using the Load Utility

The following will serve as examples for this section. The first example will load a DEL file into a table defined in a single-node nodegroup. Suppose you have a table defined in a single-node nodegroup as follows:

```
CREATE NODEGROUP ng1 on NODE(1);

CREATE TABLE branch ( Branch_ID integer not null,
                      Branch_Balance integer not null,
                      Branch_pad_0 char(40),
                      Branch_pad_1 char(40)) in ng1;
```

Since the table Branch is defined in a single nodegroup, the input file, branch.input, does not need to be split using db2split. To load the delimited input file, branch.input, you need to use the noheader option. The syntax is:

```
LOAD FROM /data1/branch.input OF DEL MODIFIED BY NOHEADER \
INSERT INTO branch
```

The second example will load ASC files with null indicators into a table defined in a single-node nodegroup. Suppose you have a table defined in a single-node nodegroup as follows:

```
CREATE NODEGROUP ng1 on NODE(1);

CREATE TABLE  fax (dept smallint,
                   deptname varchar(14),
                   manager smallint,
                   division varchar(10),
                   location varchar(13)) in ng1;
```

The file fax.asc contains the following data:

```
COL = |...+...1....+....2....+....3....+....4....+....5....+ 1
      10 Head Office   160 Corporate New York     NYNNN
      15 New England    50 Eastern   Boston       NNNyN
      20 Mid Atlantic   10 Eastern   Washington   NNyNN
      38 South Atlantic 30 Eastern   Atlanta      N?NNN
      42 Great Lakes   100 Midwest   Chicago      NNNQN
      51 Plains        140 Midwest   Dallas       NNYNN
      66 Pacific       270 Western   San FranciscoNYNNn
      84 Mountain      290 Western   Denver       NNyNN
```

1 This is not part of the data and is used to show the position of the columns.

The command is as follows:

```
db2 "LOAD FROM fax.asc OF asc MODIFIED BY NOHEADER
            METHOD L(1 2, 4 17, 19 21, 23 31, 33 45)
            NULL INDICATORS (46,47,48,49,50) REPLACE INTO fax"
```

After the load completes, the fax table contains the following:

```
DEPTNUMB   DEPTNAME       MANAGER    DIVISION     LOCATION
--------   -------------- -------    ------------ ---------------
      10   -                  160    Corporate    New York
      15   New England         50    -            Boston
      20   Mid Atlantic         -    Eastern      Washington
      38   South Atlantic      30    Eastern      Atlanta
      42   Great Lakes        100    Midwest      Chicago
      51   Plains               -    Midwest      Dallas
      66   -                  270    Western      San Francisco
      84   Mountain             -    Western      Denver
```

The next example will load DEL files which are split using db2split into a multi-node table. Suppose you have a nodegroup and table defined as follows:

```
CREATE NODEGROUP ng4 on NODES(0,1,2,3);
CREATE TABLE test1 (col1 integer,
                    col2 integer,
                    col3 integer,
                    col4 varchar(40),
                    col5 varchar(40))
                in ng4
                PARTITIONING KEY (col2) using HASHING;
```

Before loading, you need to send the split files to their corresponding hosts by using FTP. For more information on FTP, refer to 5.3.4, "Sending Partition Files to Appropriate Nodes" on page 156. In our example:

- output.00001 to directory /data1 of host1

- output.00002 to directory /data1 of host2
- output.00003 to directory /data1 of host3

The following example shows how you can load the files in parallel to the table, test1, that resides in nodegroup ng4. Nodegroup ng4 is defined on four nodes, 0-3: The steps are as follows

1. Place the load command in a file, load.test1.

```
db2 "LOAD FROM /data1/output.0000* of DEL INSERT INTO test1"
```

2. Execute the file, load.test1, in parallel using db2_all. The db2_all command is a shell script that comes with the DB2 Parallel Edition product. It executes a specified command at all DB2 logical/physical nodes specified in db2nodes.cfg.

```
db2_all ";/$HOME/load.test1" | tee load.test1.out
```

3. The above command sends the request to all the nodes specified in db2nodes.cfg. Each node executes the request in parallel. The semicolon (;) prefix inside the double quotes is required if you want the request to be executed in parallel. The messages from the load command will be sent back to the node where the db2_all is issued. The output messages will be piped to an output file, load.test1.out. The following shows the content of load.test1.out:

```
rah: host0   1
rah: host1
rah: host2
rah: host3

host0: SQL3500W  The utility is beginning the "LOAD" phase
host0: at time "10-05-1995 16:04:46.844354".   2
host0:
host0: SQL3109N  The utility is beginning to load the data from file
host0: "/data1/output.00000".      3
host0:
host0: SQL3110N  The utility has completed processing.
host0: "80000" rows were read from the input file.   4
host0:
host0: SQL3519W  Begin Load Consistency Point.  Input record
host0:           count = "80000".  5
host0:
host0: SQL3520W  Load Consistency Point was successful.  6
host0:
host0: SQL3515W  The utility has finished the "LOAD" phase
host0: at time "10-05-1995 16:05:57.486842".   7
host0:
host0: load.test1 completed ok      8

[ host1 - host3 omitted - they are identical in detail ]
```

The output from the file, load.test1.out, can be explained as follows:

1 This message indicates the command was sent to host0, host1, host2, and host3 to be executed in parallel.

2 This is an informational message indicating that the load phase is about to execute at a certain timestamp.

3 This is the normal beginning message from load. This message shows the input file for the load operation.

4 This message indicates the number of records read, and it is the normal load phase ending message.

5 The Load utility is about to perform a consistency point for the load phase.

6 The consistency point performed by load was successful.

7 This is an informational message indicating that the load phase has finished at a certain timestamp.

8 This message indicates the script, load.test1, completed execution without any errors at host0.

5.3.6.1 Considerations for Loading Data

When using load, be aware of the following:

- You cannot load into a system catalog table or a view.
- You cannot invoke load from a remote client.
- You cannot load data into a host database through DDCS.
- You cannot load into a table that has indexes. Before loading any data to a table, you must drop its indexes.
- Applications calling load must have the same code page as the database so that no data conversions will be performed.
- Load cannot process files in IXF and WSF formats.
- If there is no value in the row of the input files for a NOT NULL or NOT NULL WITH DEFAULT column, the row will be rejected since load will try to load a null value for that column if no value is given.
- Load performs best in replace mode. If you are loading into an empty table, insert will be marginally faster than replace.

5.3.7 Errors During the Load Utility

If an error occurs during the load, you can:

- Restart the load; do a load terminate followed by a load restart.
- Use load to reload the table partition on the node.
- If you are appending data to a populated table, restore the database at that node before reloading.

5.4 Autoloader Utility

A sample Autoloader Utility is provided with DB2 Parallel Edition V1.2. The Autoloader utility consists of a program and a set of shell scripts. It is used to:

- Transfer data from one local or remote source system (MVS, VM, or RS/6000) to AIX system (RS/6000, SP) using FTP.
- Partition that data using db2split.

- Load the split data to the corresponding nodes using the load utility.

Autoloader uses pipes between the FTP, db2split, and load so the data movement from the source to target systems is automated. This eliminates the need for temporary spaces to hold staging data for FTP and db2split.

For more information about Autoloader, refer to the README file in /usr/lpp/db2pe_01_02/samples/autoloader or *Migrating and Managing Data on RS/6000 SP with DB2 Parallel Edition, SG24-4658*.

5.4.1 Considerations for the Autoloader Utility

There are a number of steps to consider before running Autoloader.

1. Create a directory on a file system which is shared across nodes via NFS. Copy the contents of $HOME/sqllib/samples/autoloader into this directory.
2. The Autoloader script uses three programs: db2, db2split and mknod. Check what directory these programs reside in on your machine using the following commands:

   ```
   whence -v db2

   whence -v db2split

   whence -v mknod
   ```

   Modify the Autoloader script to ensure that the shell variables DB2_DIR, DB2SPLIT_DIR and MKNOD_DIR point to the directories that you found using the above whence commands.
3. Make sure that the script rksh is in this directory.
4. If the module channel is not there, compile the channel.c program using the C compile.

   ```
   cc -o channel channel.c
   ```
5. You will need to add the userid and password information in the following file if your source file resides on a remote system. Put in $HOME/.netrc the following information:

   ```
   machine <machine_name> login <login_name> password <password>
   ```

   machine_name hostname of the source system where the source file resides

   login_name login name of your userid on the source system

   password password of your userid on the source system
6. Ensure that the permission of .netrc files are -rw-------. If not, use the following command to change the permission:

   ```
   chmod 600 .netrc
   ```
7. Customize your Autoloader specification file. For more information, see 5.4.2, “Customize the Autoloader Specification File” on page 167.
8. Edit the db2split configuration files and modify according to your configuration. In the case of non-delimited data make sure that the field RecLen is exactly equal to the number of columns in the

data file (excluding end of line character). Modify the parameter Nodes according to your database configuration. Specify every node in the OutputNodes parameter. Otherwise, AutoLoader takes all the nodes as default value. If you want the table to span all the nodes, you need not specify OutputNodes. If you are using local files, place the appropriate file name in the InFile parameter. For non-local files, the directory name for the input files is read from the spec_file.

9. Customize your db2split configuration file. Refer to the *DB2 Parallel Edition Administration Guide and Reference* for the list of parameters you can specify in the configuration file. The following shows the steps to consider when customizing your db2split configuration file for Autoloader:

   a. If the input data is non-delimited, you must make sure that the parameter RecLen is exactly equal to the number of columns in the data file (excluding end of line character).

   b. Modify the parameter Nodes according to your database configuration. Instead of using the Nodes parameter, you can use Mapfili to specify an input partitioning map.

   c. Specify every node in the OutputNodes parameter where the data is going to be populated. Otherwise, Autoloader will takes the nodes from the db2nodes.cfg file as default.

10. If the input data is non-delimited, you must create a load script. For more information, see 5.4.3, “Load Script File” on page 167.

11. Make sure the database and tables where the data is split and loaded by Autoloader already exist.

12. Make sure you have enough maximum number of concurrent applications. (MAXAPPLS). You can change MAXAPPLS by updating the database configuration. The following shows an example of how to change the MAXAPPLS for DSS database to 100:

```
db2_all ";db2 update database configuration for DSS \
using maxappls 100"
```

After running the Autoloader, the following steps are recommended:

1. Check the file autoloader.log, which contains messages from the nodes that have completed the load process. If you are loading M tables on N nodes, you will find M * N entries in autoloader.log file.

2. Check the file <table_name>.log which contains messages from the split process in Autoloader.

3. After Autoloader completes successfully, check the row-count of the tables by issuing the command select count(*) against the table.

4. Run the cleanup program to clean up all the temporary files and named pipes (FIFOs).

5. If the Autoloader operations have completed and the row-count of the tables is correct, back up the database before using.

Note: After running Autoloader, if the process is hanging without any CPU and I/O usage, you should check the autoloader.log and <table.name>.log for any error. You may also have to cancel the Autoloader process by entering Cntrl c and then run cleanup to kill all the Autoloader processes, and clean up all the temporary files and named pipes (FIFOs).

5.4.2 Customize the Autoloader Specification File

The Autoloader specification file is used to provide Autoloader with the information needed to transfer, split, and load the input data from a local or remote system to the database on AIX. The specification file contains the following format:

```
<directory_name on source_system>
<file_name 1> <tab_name> <db2split cfg file> <load script>
<file_name 2> <tab_name> <db2split cfg file> <load script>
      :            :               :          :
<file_name n> <tab_name> <db2split cfg file> <load script>
```

<directory_name on source_system> The directory where the file to be loaded exists. If the source system is MVS or VM, then leave the first line blank.

<file_name x> The source file that you want to split and load.

<table_name> The table name in the database where the data is loaded into.

<cfg file used by db2split> The name of the configuration file to be used by db2split.

<load script> The name of the load script file. This is used only if you are using non-delimited ASCII input data. For the format of the load script file, refer to 5.4.3, “Load Script File.”

5.4.3 Load Script File

When your input data is a non-delimited file, you must create a load script file to specify the starting and ending columns of the input using METHOD L parameter of the load command. A sample load script ″sample.load″ has been provided for you and can be found in the directory /usr/lpp/db2pe_01_02/samples/autoloader. You only need to modify the column positions in the METHOD L parameter of the load command. Autoloader will automatically update the values for mydatabase, inputfile, and mytable.

The following is the content of the ″sample.load″ file:

```
db2 connect to mydatabase
db2 "load from inputfile of asc method l (1 4,6 10,12 12,14 18) replace
    into mytable"
db2 connect reset
```

For more information about the load command, refer to 5.3.5, "Load Utility" on page 157 or *DB2 Parallel Edition for AIX Administration Guide and Reference*

5.4.4 Using AUTOLOADER

All the shell scripts and program for Autoloader can be found in the directory /usr/lpp/db2pe_01_02/samples/autoloader. It consists of the following:

README The Autoloader README file

autoloader The main Autoloader driver

cleanup The shell script file to clean up the temporary files and named pipes (FIFOs) and processes created during the execution of Autoloader

channel.c The C source file to read the data from stdin and write to a named pipe (FIFO)

channel The module of channel.c. If it is not in the directory, you will need to compile the channel.c program using the C compiler in your system

rksh A shell script file used to simulate the semantics of rsh

sample.spec A sample Autoloader specification file

sample.load A sample load script file for non-delimited input data

The following is the syntax:

```
autoloader [-d] [-c coldel] [-h host_name]
[-s spec_file] database_name
```

where the options are:

-d Indicates delimited data will be used. The default is non-delimited data.

-c Indicates a column delimiter other than the default delimiter '|' will be used.

-h Indicates the input files are from a remote system with the specified "host_name". Omit this option if the input files are from local system.

-s Indicates the specification file name will be used. The default is "sample.spec".

Note: Make sure to execute the cleanup program when the Autoloader operation completes either with or without error. This will clean up all the temporary files and named pipes (FIFOs) which were created during the Autoloader operation.

5.4.5 Autoloader Process

The following describes the steps involved in the Autoloader operation:

1. Process the `autoloader` command line options.
2. Log the input options in a file called 'input.log'.
3. If the '-h' option is specified, call the function `generate_transfer` in the Autoloader script to prepare the FTP scripts for each file in the Autoloader specification file.
4. On each node specified in db2nodes.cfg, create a directory called /tmp/$whoami.autoloader. The $whoami will be the userid used to execute autoloader.
5. Create the named pipes (FIFOs) for each file specified in the Autoloader specification file.
6. Obtain the table name, the name of the db2split configuration file, and the name of the load script file from Autoloader specification file.
7. Call function `modify_cfg` in the Autoloader script to set the following parameters in the db2split configuration file:
   - Description to the database name
   - InFile to the named pipe (FIFO) created in previous step if the '-h' option is specified.
   - LogFile to table_name.log,w
   - OutFile to table_name
   - CDelimiter to the column delimiter
8. Call function `start_load` in Autoloader script to:
   a. Generate the named pipes (FIFOs) for data being transferred via FTP and for data being split.
   b. If the input data is non-delimited, call function `create_dbload_file` to modify the load script specified in the Autoloader specification file to replace:
      1) The value of mydatabase with the name of the database where the split data will be loaded.
      2) The value of inputfile with the split named pipes (FIFOs).
      3) The value of mytable with the name of the table where split data will be loaded to.
      4) The value of msgfile with table_name.msg.
   c. Initiate the splitting (db2split) and loading (db2load) processes.

   When the parameter OutputNodes in the splitter configuration file is specified, Autoloader executes the steps above for each node in OutputNodes. Otherwise, each node in the db2nodes.cfg will be used.
9. If the '-h' option is specified, start FTPing the input file from remote system.

5.4.6 Performance Considerations

Since the Autoloader combines the split and load processes, the speed that can be achieved by Autoloader is limited to the slowest component in the process. The following shows the rate of the load and split:

- load rate : about 1.5 to 1.8G/hour/node
- split rate : 1 G/hour

Generally, when the input data is on a disk, the split process usually creates the bottleneck. Even though the load process can achieve a rate of 1.5 GB to 1.8 GB/hour/node, it will be blocked on the named pipes (FIFOs) for most of the time. To resolve the bottleneck problem, you will have to find a way to split faster so that all the load processes are busy. One way of addressing this issue is to have a node dedicated to splitting the data.

5.4.7 Example Using Autoloader Utility

The following will serve as examples for this section. The first example will autoload a delimited input file into a table defined in a multi-node nodegroup. Suppose you have a nodegroup and table defined as follows:

```
CREATE NODEGROUP order_line on NODES(0,1,2,3);
CREATE TABLE dss.orders(O_ORDERKEY INTEGER NOT NULL,
                        O_CUSTKEY INTEGER NOT NULL,
                        O_ORDERSTATUS CHAR(1) NOT NULL,
                        O_TOTALPRICE decimal(10,2) NOT NULL,
                        O_ORDERDATE DATE NOT NULL,
                        O_ORDERPRIORITY CHAR(15) NOT NULL,
                        O_CLERK CHAR(15) NOT NULL,
                        O_SHIPPRIORITY INTEGER NOT NULL,
                        O_COMMENT VARCHAR(79) NOT NULL)
                      IN order_line
               PARTITIONING KEY (O_ORDERKEY);
```

The following shows the steps to load the data from input file order.tbl residing in the directory /data to dss.orders table in the database dss using Autoloader.

1. Customize the splitter configuration file orders.cfg. The following shows its content:

```
Description=orders
InFile=/data/order.tbl
RecLen=32000
Nodes=(0,1,2,3)
OutputNodes=(0,1,2,3)
LogFile=Log,w
OutFile=orders
CDelimiter=|
RunType= PARTITION
Partition=o_orderkey,1,,,NN,INTEGER
Trace=0
```

2. Customize the Autoloader specification file orders.spec. The following shows its content:

```
/data
order.tbl dss.orders /u/db2pe/autoloader/orders.cfg
```

3. Execute the Autoloader using the following command:

```
autoloader -d -s orders.spec dss
```

4. After running Autoloader, perform the following steps:

   a. Check the contents of the following files:

      1) autoloader.log

      The following messages were produced in the log:

```
Loading dss.orders.00000 in dss.orders on db2pe
        Over with Messages
  SQL authorization ID   = DB2PE
SQL3500W  The utility is beginning the
          "LOAD" phase at time "09-02-1996
SQL3109N  The utility is beginning to load the data from file
SQL3110N  The utility has completed processing.
          "37632" rows were read from
SQL3519W  Begin Load Consistency Point.
          Input record count = "37632".
SQL3520W  Load Consistency Point was successful.
SQL3515W  The utility has finished the
          "LOAD" phase at time "09-02-1996
DB20000I  The SQL command completed successfully.

[ Messages for dss.orders.00001 - dss.orders.00003 omitted -
they are identical in detail ]
```

      2) dss.orders.log

      The following messages were produced in the log:

```
dss>Log file opened successfully
dss> Start time: Mon Sep 02 15:56:09 1996
dss> Input file /data/order.tbl opened successfully
dss> Input maximum record length :32000
dss> Program is running with CHECK level
dss> The string delimiter is :<">
dss> Tracing 0 delimited (delimiter <|>) record(s)
dss> Getting partitioning map...done
dss> The Run Type is PARTITION
dss> Output partitioning map file not used
dss> The message level is NOWARN
dss> Distribution file name: DISTFILE
dss> Distribution file DISTFILE opened successfully
     for writing
dss> Working on  1 keys.
dss> o_orderkey Start:    0  Len:    0  Position:  1
```

```
                    Type: NN INTEGER
dss> Output files will be dss.orders.00xxx
dss> All output files opened successfully
dss> Processed        50000
dss> Processed       100000
dss> Processed       150000
dss> Writing distribution map to DISTFILE
dss> Total record count:    150000
dss> Total record discarded:         0
dss> Stop time: Mon Sep 02 16:00:48 1996
dss> Elapsed time:  0 hours,  4 minutes, 39 seconds
dss> Throughput: 537 records/sec
dss> Record counts for output nodes:
Node: 3: Record count: 37193
Node: 2: Record count: 37569
Node: 1: Record count: 37606
Node: 0: Record count: 37632
dss> Complete.
Program ran successfully with 0 warning message(s) and 0
discarding record(s)
```

b. Clean up all the temporary files and named pipes by issuing the following command:

```
cleanup
```

5. After Autoloader completes successfully, check the row count of dss.orders table by issuing:

```
db2 connect to dss
db2 "select count(*) from dss.orders"
db2 connect reset
```

The next example will autoload a non-delimited input file into a table defined in a multi-node nodegroup. Suppose you have a nodegroup and table defined as follows:

```
CREATE NODEGROUP part_partsupp on NODES(0,1,2,3);
CREATE TABLE dss.part (P_PARTKEY    INTEGER          NOT NULL,
                       P_NAME       VARCHAR  (55)    NOT NULL,
                       P_MFGR       CHAR     (25)    NOT NULL,
                       P_BRAND      CHAR     (10)    NOT NULL,
                       P_TYPE       VARCHAR  (25)    NOT NULL,
                       P_SIZE       INTEGER          NOT NULL,
                       P_CONTAINER  CHAR     (10)    NOT NULL,
                       P_RETAILPRICE decimal(10,2)   NOT NULL,
                       P_COMMENT    VARCHAR  (23)    NOT NULL)
                      IN part_partsupp
             PARTITIONING KEY (P_PARTKEY);
```

The following shows the steps to load the non-delimited data from input file part.tbl residing in the directory /data to dss.part table in the database dss using Autoloader.

1. Obtain the partitioning map of the nodegroup part_partsupp by issuing:

```
db2gpmap -d dss -m part_partsupp.pmap -g part_partsupp
```

2. Customize the splitter configuration file part.cfg. The following shows its content:

```
Description=part
InFile=/data/part.tbl
RecLen=170
OutputNodes=(0,1,2,3)
mapfili=part_partsupp.pmap
LogFile=Log,w
OutFile=part
RunType= PARTITION
Partition=p_partkey,1,1,4,NN,INTEGER
Trace=0
```

3. Customize the load script part.load. The following shows its content:

```
$HOME/sqllib/bin/db2 connect to mydatabase
$HOME/sqllib/bin/db2 "load from inputfile of asc method l
(1 4, 6 60, 62 86, 88 97, 99 123, 125 126, 128 137, 139 146, 148 170)
replace into mytable"
$HOME/sqllib/bin/db2 connect reset
```

4. Customize the Autoloader specification file part.spec. The following shows its content:

```
/data
part.tbl dss.part part.cfg part.load
```

5. After running Autoloader, perform the following steps:

   a. Check the contents of the following files:

      1) autoloader.log

         The following messages were produced in the log:

```
Loading dss.part.00000 in dss.part on db2pe Over with Messages
  SQL authorization ID   = DB2PE
SQL3500W  The utility is beginning the
          "LOAD" phase at time "09-02-1996
SQL3519W  Begin Load Consistency Point.
          Input record count = "0".
SQL3520W  Load Consistency Point was successful.
SQL3109N  The utility is beginning to load the data from file
SQL3110N  The utility has completed processing.
          "20" rows were read from the
SQL3519W  Begin Load Consistency Point.
          Input record count = "20".
SQL3520W  Load Consistency Point was successful.
SQL3515W  The utility has finished the
          "LOAD" phase at time "09-02-1996
DB20000I  The SQL command completed successfully.
```

```
[ Messages for dss.part.00001 - dss.part.00003 omitted -
they are identical in detail ]
```

2) dss.part.log

The following messages were produced in the log:

```
dss>Log file opened successfully
dss> Start time: Mon Sep 02 17:46:38 1996
dss> Input file /data/part.tbl opened successfully
dss> Input maximum record length :170
dss> Program is running with CHECK level
dss> Tracing 0 non-delimited record(s)
dss> Input map file part_partsupp.pmap opened
     successfully for reading
dss> Getting partitioning map...done
dss> The Run Type is PARTITION
dss> Output partitioning map file not used
dss> The message level is NOWARN
dss> Distribution file name: DISTFILE
dss> Distribution file DISTFILE opened successfully
     for writing
dss> Working on  1 keys.
dss> p_partkey     Start: 1  Len: 4  Position: 1
     Type: NN INTEGER
dss> Output files will be dss.part.00xxx
dss> All output files opened successfully
dss> Writing distribution map to DISTFILE
dss> Total record count:        80
dss> Total record discarded:         0
dss> Stop time: Mon Sep 02 17:46:46 1996
dss> Elapsed time:  0 hours,  0 minutes,  8 seconds
dss> Throughput: 10 records/sec
dss> Record counts for output nodes:
Node: 3: Record count: 20
Node: 2: Record count: 20
Node: 1: Record count: 20
Node: 0: Record count: 20
dss> Complete.
Program ran successfully with 0 warning message(s) and 0
discarding record(s)
```

b. Clean up all the temporary files and named pipes by issuing:

```
cleanup
```

6. After autoloader completed successfully, check the row count of dss.part by issuing:

```
db2 connect to dss
db2 "select count(*) from dss.part"
db2 connect reset
```

Note: There is an error in the autoloader script to handle logical nodes when autoloading from a non-delimited input file to a table

spread on multiple logical nodes. Make sure the export DB2NODE=$nodenum statement exists in the assignment of stmt3 variable in the Autoloader script.

5.5 Import/Export Utilities

The import/export utilities are used to move data between databases. The Import utility is used to insert data from an input file into a table or view. The authorization for Import in DB2 Parallel Edition is the same as it is in DB2/6000 Version 1.

5.5.1 Using the Import Utility

The syntax of the Import utility in DB2 Parallel Edition is the same as it is in DB2/6000 Version 1. For more information about the Import utility, see the *DB2/6000 Command Reference*.

The following are considerations for the Import utility:

- The Import utility does not use the buffered insert feature.
- If the table is defined in a multi-node nodegroup, the replace option will delete all existing data in the table partition at each node in parallel.
- To delete all rows from a large table, the import utility with an empty input file can be used to perform the delete process faster than using the delete SQL statement:

```
db2 "IMPORT FROM /dev/null OF DEL MODIFIED BY COLDEL \
     REPLACE INTO test2"
```

5.5.2 Using the Export Utility

The Export utility is used to export data from a table or view into a file. The authorization and syntax for the Export utility are the same as they are in DB2/6000 Version 1. For more information about the Export utility, see the *DB2/6000 Command Reference*.

In Parallel Edition, you can run the Export utility to have each node perform an export locally by using the NODENUMBER and CURRENT NODE as predicates of the selection condition. For more information about NODENUMBER and CURRENT NODE, see 4.4.6, "SQL Functions" on page 87.

5.5.3 Executing the Export Utility in Parallel

The following example shows how to export in parallel from the table, test1, defined in a four-node nodegroup to a file called test1.exp.o using a delimited format.

1. Place the export command in a file called export.test1.

```
db2 "EXPORT to /data1/test1.exp.o of DEL
 SELECT * FROM test1 WHERE NODENUMBER(col1) = CURRENT NODE"
```

2. Propagate the export command in the export.test1 file to all nodes by using db2_all.

```
db2_all ";$HOME/export.test1" | tee export.test1.out
```

3. The messages from the export command will be sent back to the node where the db2_all is issued. The output messages will be piped to the file called export.test1.out. The following shows the content of the export.test1.out file:

```
rah: host0      1
rah: host1
rah: host2
rah: host3

host0: SQL3104N  The Export utility is beginning to
host0: export data to file "/data1/test1.exp.o".    2
host0:
host0: SQL3105N  The Export utility has finished
host0: exporting "80000" rows.                      3
host0:
host0: export.test1 completed ok      4

[ host1 - host3 omitted - they are identical in detail ]
```

1 This message indicates the command was sent to host0, host1, host2, and host3 to execute in parallel.

2 This is an informational message indicating the export utility has begun exporting data to the file. In this example, test1.exp.o is the output file, and it resides in the /data1 directory.

3 This message indicates the number of rows that have been exported.

4 This message indicates the script, export.test1, completed the execution on host0 without encountering any errors.

5.5.4 File Formats for Import/Export

The following are valid file formats for the import and export utilities:

DEL With delimited ASCII, data values are separated by a special delimiting character. This character must only be used for this purpose within the file. Each line of the file contains the data for a row of the table.

ASC Non-delimited ASCII requires the data to be aligned to columns. If the file contains data for a row of the table, and specific columns within a line are associated with each field, ASC may not be used with the export utility.

WSF Work-Sheet Format is used when transferring data to or from work-sheet products.

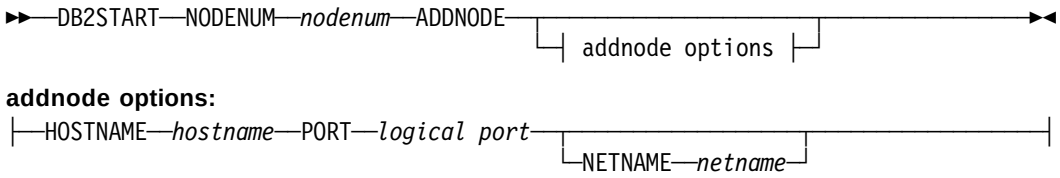
IXF This is the PC version of the Integrated Exchange Format.

5.6 Adding Nodes

DB2 Parallel Edition provides tools to scale your configuration by adding a new node to your parallel database system. You can add nodes to the system either when the DB2 Parallel Edition database manager is running or when it is stopped. This section will describe the process of adding a node to a parallel database system with the assumption that the node to be added has been installed and configured to both the parallel database and to the operating system. In our examples, we will show only adding the node to the parallel database system. To perform the add node operation, you must have SYSADM authority.

There are two methods used to add a node to a system. The first method uses the db2start command with the addnode option. The syntax is as follows:

DB2START ADDNODE Command



The following parameters are described in detail:

nodenum This parameter is the node number assigned to the new node.

hostname This parameter is the host name to be added to the $HOME/sqllib/db2nodes.cfg file.

logical port This parameter is the logical port to be added to the $HOME/sqllib/db2nodes.cfg file. Valid values are from 0 to 999.

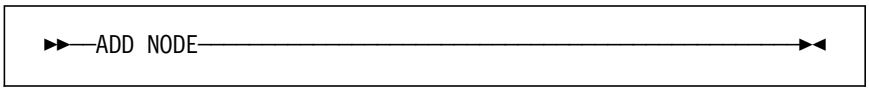
netname This parameter is optional and is used to support a host that has more than one active TCP/IP interface, each with its own hostname. If a switch network is installed in a RS/6000 SP machines, the switch name will be used. If you do not specify a value, this parameter defaults to the value specified for hostname.

For a description of all of the parameters, see the *DB2 Parallel Edition for AIX Administration Guide and Reference*.

Using the db2start command with the addnode option will:

- Add the new node to $HOME/sqllib/db2nodes.cfg file.
- Export the DB2NODE environment variable using the specified nodenum.
- Create the database partition on the new node for every database that already exists in the system. (The partition remains empty until you move data to it using redistribute nodegroup).

The second method to adding a node to a parallel database system is by using the add node command. The syntax is as follows:



The add node command must be issued from the node being added to the parallel system. The add node command will:

- Create the database partition on the new node for every database that already exists in the system. (The partition remains empty until you move data to it using redistribute nodegroup).

The following are considerations when adding nodes to your parallel database system:

- You must ensure the processor on which the new node is being added has enough disk space so that segment directories can be created for all existing databases on the system.
- You should not attempt to create/drop a database while the add node operation is in progress.
- Data will not be moved to the newly added node until you perform the redistribute nodegroup command.
- It is recommended that you back up all of the databases on the new node.

5.6.1 Adding a Node When the Database Manager is Active

The following are the steps to add a node to a parallel database system while the database manager is running. The newly added node does not become available to databases until the database manager is shut down and restarted. Before proceeding, you must ensure that the node to be added has been installed and configured to the system.

1. Issue the db2start command with the addnode option from any node.
2. When the system is ready to shutdown, stop the database manager by issuing the db2stop command. After all the nodes in the system are stopped, the $HOME/sqllib/db2nodes.cfg file will be updated to include the new node.
3. Restart the system by issuing a db2start command.

5.6.2 Adding a Node When the Database Manager is Inactive

The following steps are necessary to add a node to a parallel database system in which the database manager has been stopped. Before proceeding, you must ensure that the node to be added has been installed and configured to the system.

1. Edit the $HOME/sqllib/db2nodes.cfg file to include the new node.
2. Issue the following command to start the database manager on the new node:

```
db2start NODENUM nodenum
```

3. If the new node is a logical node, export the DB2NODE environment variable with the node number to be added.

```
export DB2NODE=nodenum
#where nodenum is the number of the added node
```

4. Run the Add Node utility on the new node:

```
db2 add node
```

5. Start the database manager on other nodes by issuing the db2start command.

5.6.3 Add Node Example

This example shows how to add a logical node to a parallel database system which already has three physical nodes. There is one database distributed across the three nodes. The entries in the $HOME/sqllib/db2nodes.cfg file before adding the node are:

```
0  HOST0   0
1  HOST1   0
2  HOST2   0
```

The following are the steps to add a logical node to the system in which the database manager is already stopped:

1. Edit the $HOME/sqllib/db2nodes.cfg file to add the entry

```
3  HOST2   1
```

2. Log onto HOST2, and issue the following command to start the database manager for node number 3.

```
db2start nodenum 3
```

3. Execute a shell script containing the following:

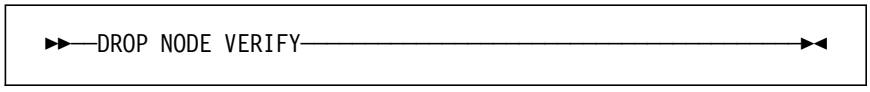
```
export DB2NODE=3
db2 ADD NODE
```

4. Issue db2start to start the remaining the nodes in the system.

5.7 Dropping Nodes

DB2 Parallel Edition provides tools to scale your configuration by dropping a node from your parallel database system. This section will describe that process. To perform the drop node operation, you must have SYSADM authority.

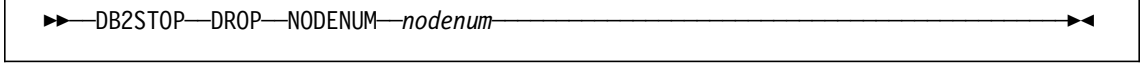
You can use the drop node verify command to check whether the node to be dropped is being used by any database. The syntax is as follows:



Note: Do not forget to REDISTRIBUTE all data on the node that you want to drop.

To drop the node, issue the db2stop command with the drop nodenum option. The syntax is as follows:

DB2STOP DROP NODENUM Command



Issuing db2stop with the drop nodenum option will:

- Stop all the nodes in the $HOME/sqllib/db2nodes.cfg file.
- Clean up the segment subdirectories for the node being dropped.
- Remove the entry in the $HOME/sqllib/db2nodes.cfg file for the node being dropped.

5.7.1 Dropping a Node When the Database Manager is Active

The following steps are necessary to drop a node from a parallel database system where the database manager is active. Before proceeding, if data exists on the node being dropped, you must redistribute the data that resides on this node for every database to ensure the partitioning map is kept current.

1. Issue the drop node verify command on the node to be dropped to to verify that the node is not in use.
   - If message SQL6034W is received, you can proceed with the drop node process.
   - If message SQL6035W is received, you must use the redistribute nodegroup command to move the data from the node being dropped to other nodes of the database. You cannot drop the node until this is completed.
2. Issue the db2stop command with the drop nodenum option to drop the node.
3. Restart the database manager by issuing the db2start command.

5.7.2 Drop Node Example

This example shows how to drop a logical node from a parallel database system. The entries in the $HOME/sqllib/db2nodes.cfg file before dropping the node are:

```
0  HOST0   0
1  HOST1   0
2  HOST2   0
3  HOST2   1
```

The following steps will drop a node (number 3) from the system:

1. Issue the following shell script to verify if node 3 is in use:

   ```
   export DB2NODE=3
   db2 drop node verify
   ```

   The following message is returned:

   ```
   SQL6034W NODE "3" is not being used by any databases
   ```

2. Issue the following command to drop the node number 3 from the system:

   ```
   DB2STOP DROP NODENUM 3
   ```

3. Issue db2start to restart the database manager.

5.7.3 Data Redistribution

The Redistribute Nodegroup utility is used to redistribute data among the nodes in an existing nodegroup. This section provides detail about the data redistribution process and how you can use the redistribution utility to perform the following tasks:

- Node redistributing
- Adding nodes to a nodegroup
- Dropping nodes from a nodegroup

Data redistribution is done at the nodegroup level rather than at the table level. The set of tables in the specified nodegroup of a database will be affected when a redistribution operation is done on that nodegroup. The following three system catalog tables are used and modified during the redistribution processes:

- **SYSIBM.SYSPARTITIONMAPS**

  This catalog table contains information for each partitioning map that is created.

  Table 8 contains a description of the columns in SYSIBM.SYSPARTITIONMAPS.

Table 8. SYSIBM.SYSPARTITIONMAPS Catalog Table

Column Name	Data Type	Description
PMAP_ID	SMALLINT	Internal partitioning map ID.
PARTITIONMAP	LONG VARCHAR	Partitioning map. Partitioning maps are described in 3.7.6, “Partitioning Map” on page 67. This contains 4096 values for a multi-node nodegroup and one value for a single-node nodegroup.

- **SYSIBM.SYSNODEGROUPS**

  This catalog table contains information for each nodegroup that is created. At database creation time, two system nodegroups (IBMCATGROUP and IBMDEFAULTGROUP) are created.

  Table 9 contains a description of the columns in SYSIBM.SYSNODEGROUPS.

Table 9 (Page 1 of 2). SYSIBM.SYSNODEGROUPS Catalog Table

Column Name	Data Type	Description
NAME	VARCHAR(18)	Name of the nodegroup.
DEFINER	CHAR(8)	Authorization ID of the user that defined the nodegroup.
PMAP_ID	SMALLINT	Internal partitioning map ID. This references an existing partitioning map record in the SYSIBM.SYSPARTITIONMAPS catalog table.

Table 9 (Page 2 of 2). SYSIBM.SYSNODEGROUPS Catalog Table		
Column Name	**Data Type**	**Description**
REBALANCE_PMAP_ID	SMALLINT	Internal partitioning map ID created during data redistribution. If REBALANCE_PMAP_ID is -1, data redistribution is not taking place. If it contains any other value, REBALANCE_PMAP_ID references an existing partitioning map record in the SYSIBM.SYSPARTITIONMAPS catalog table. After data redistribution, the content of REBALANCE_PMAP_ID will be copied to PMAP_ID, and the value will then be set to -1.
CTIME	TIMESTAMP	Creation timestamp.
REMARKS	VARCHAR(254)	User-provided comment. Reserved for future implementation.

- **SYSIBM.SYSNODEGROUPDEF**

  This catalog table contains information for the many-to-many relationship between nodes and nodegroups. A record is inserted for each node that a nodegroup is defined on.

  Table 10 contains a description of the columns in SYSIBM.SYSNODEGROUPDEF.

Table 10. SYSIBM.SYSNODEGROUPDEF Catalog Table		
Column Name	**Data Type**	**Description**
NGNAME	VARCHAR(18)	Name of the nodegroup. NGNAME references an existing nodegroup in SYSNODEGROUPS.
NODENUM	SMALLINT	The corresponding node number of the node. The node number is declared in the node configuration file, db2nodes.cfg. See 3.3, “Parallel Database Nodes” on page 45 for more details. The node number must be in the range 0 to 999.
NODENAME	VARCHAR(18)	A node name created by concatenating the string ″NODE″ and the node number. For example, ″NODE00000″.
IN_USE	CHAR(1)	Whether the node number is in the partitioning map of the nodegroup: Y = node is in the partitioning map. D = node to be dropped from partitioning map after current data redistribution operation completes successfully.

5.7.4 Redistribution Process

The following describes the steps done on the nodegroup by the data redistribution operation in DB2 Parallel Edition:

1. Obtain a new partitioning map ID (PMAP_ID) for the target partitioning map and insert into SYSPARTITIONMAPS catalog table.

2. Update the REBALANCE_PMAP_ID column of the record in SYSIBM.SYSNODEGROUPS for the nodegroup being processed using the newly obtained PMAP_ID.
3. Add lines for new nodes in the nodegroup, if any, to SYSNODEGROUPDEF.
4. Adds any new nodes in the nodegroup, if any, to SYSNODEGROUPDEF.
5. Set the IN_USE column in SYSNODEGROUPDEF to ′D′ for any node that is to be dropped.
6. COMMIT the catalog table updates.
7. For all new nodes being added, create database files.
8. For each table in the nodegroup, redistribute the data in table, committing after each one.
9. For nodes being explicitly or implicitly dropped, delete database files, and delete entries in SYSIBM.SYSNODEGROUPDEF.
10. Update the nodegroup record in SYSIBM.SYSNODEGROUPS to set PMAP_ID = REBALANCE_PMAP_ID, and REBLANCE_PMAP_ID = -1.
11. Delete the old partitioning map from SYSIBM.SYSPARTITIONMAPS.
12. COMMIT the catalog table updates.

5.7.5 Redistributing Data on Each Table

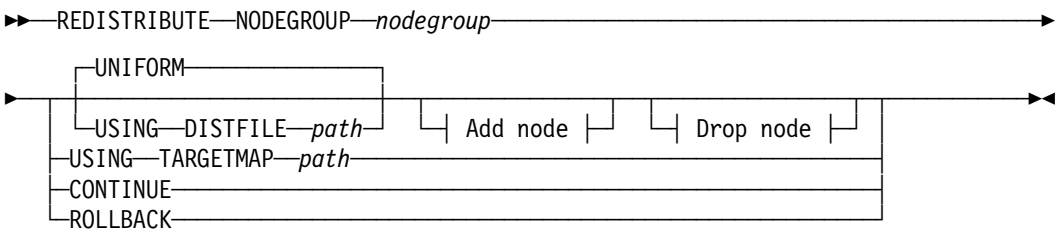
The redistribution executes on each table successively for all the tables in the specific nodegroup. For each table, the following steps are executed:

1. Lock the table in exclusive mode.
2. Invalidate all plans involving this table. The PMAP_ID associated with the table will change since the table is being redistributed. Invalidating plans will force the compiler to obtain the new partitioning information for the table and to generate plans accordingly.
3. Perform data redistribution of the table.
4. If the redistribution operation was successful, then:
   a. Issue a commit for the table.
   b. Release the table lock.
   c. Continue with next table in the nodegroup
5. If the operation failed before the table was fully redistributed, then:
   a. Rollback updates to the table.
   b. Release the table lock.
   c. Terminate the entire redistribution operation and return with an error.

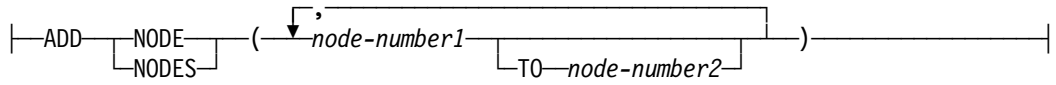
5.7.6 Redistribute Utility

The syntax for the Redistribute Nodegroup utility is as follows:

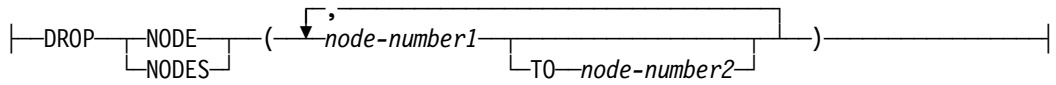
Redistribute Nodegroup Command/API



Add node clause:



Drop node clause:



Note:

- Tables in the system nodegroup, IBMCATGROUP, cannot be redistributed.
- The utility must be executed on the catalog node.
- The nodegroup specified must exist.
- Multiple copies of the Redistribute Nodegroup utility cannot be executed concurrently against the same nodegroup.
- Make sure your log file size is large enough for the insert and delete operations done at each node during the redistribution process.
- The utility may execute faster and use less log space if indices are dropped first. Index maintenance can require large amounts of log space as leaf pages are split during insertions. If indices are not dropped before redistribution, table reorganization may be required for optimal performance.
- Make sure the nodes to be added are defined in the parallel database system. The nodes listed must be unique and cannot appear in the drop clause.
- Make sure the nodes to be dropped are the members of the nodegroup specified. The nodes listed must be unique and cannot appear in the add clause.

5.7.7 Node Redistribution

There are different ways to perform node redistribution using the Redistribute Nodegroup utility. You can:

1. Distribute the data uniformly across all the nodes of the nodegroup.
2. Distribute the data using a target partitioning map.
3. Distribute the data using a distribution file.

5.7.7.1 Uniform Data Distribution

By default, the redistribute nodegroup utility assumes that each of the 4096 hash partitions represents the same weight or the same amount of data, and the hash partitions are uniformly distributed across all nodes in the nodegroup.

This can be useful if you know that your tables in the nodegroup have approximately the same number of rows that will hash to each hash partition.

5.7.7.2 Data Distribution Using a Target Partitioning Map

The redistribute nodegroup utility uses a target partitioning map to do the data redistribution. You provide a target map as an input to the utility with the using targetmap option. The partitioning map you provide can be created by db2split program using analyze mode based on the data distribution of a particular table in the nodegroup, or it can be created by the user. For more information about db2split, see 5.3.2, “Partitioning Data with db2split” on page 150. The user-defined partitioning map can be derived using the partition and nodenumber SQL functions. For more information about SQL functions, see 4.4.6, “SQL Functions” on page 87.

The target partitioning map must contain:

- 4096 entries if the resultant nodegroup is a multi-node nodegroup.
- 1 entry if the resultant nodegroup is a single-node nodegroup.

5.7.7.3 Redistribution Example Using a Target Partitioning Map

The following example shows how to redistribute the data in a 4-node nodegroup, ng4, using a target partitioning map, ng4.pmap, which is created by db2split using analyze mode:

```
REDISTRIBUTE NODEGROUP ng4 USING TARGETMAP $HOME/split/ng4.pmap
```

5.7.7.4 Data Distribution Using a Distribution File

If your data is skewed, you can provide a distribution file as an input to the redistribution utility with the USING DISTFILE option to achieve even data redistribution across all nodes in the nodegroup.

The distribution file contains an integer value for each of the 4096 hash partitions. The distribution file is created whenever the db2split program splits a table. You can use the partition and nodenumber SQL functions to find out the current data distribution across partitions and nodes. Then use this information to derive a distribution file. For more information about SQL functions, see 4.4.6, “SQL Functions” on page 87.

5.7.7.5 Example of Data Distribution Using a Distribution File

The following example shows how to redistribute the data in a 4-node nodegroup, ng4, using a distribution file, ng4.distfile, which is created by db2split:

```
REDISTRIBUTE NODEGROUP ng4 USING DISTFILE $HOME/split/ng4.distfile
```

5.7.8 Adding Nodes to a Nodegroup

You can use the redistribute nodegroup utility to explicitly or implicitly add nodes to a nodegroup. The nodes added must already be defined in the parallel database system. Therefore, entries for the nodes will be added in db2nodes.cfg. To add nodes implicitly, you use a target partitioning map which includes the node number you want to add.

When you add nodes to a nodegroup, the utility will:

- Redistribute the data from other nodes in the nodegroup to the nodes which are being added to the nodegroup.
- Update the SYSIBM.SYSNODEGROUPDEF catalog table.
- Generate a new partitioning map for the nodegroup if nodes are added explicitly.
- Catalog the target partitioning map as the new partitioning map for the nodegroup if nodes are added implicitly.

5.7.8.1 Examples of Adding a Node

The section will show examples of adding a node either explicitly or implicitly to a nodegroup.

The following example shows how to use the redistribute nodegroup utility to add one node (node 1) explicitly in the nodegroup, NG, which is created on nodes 0 and 2. The table, ORDERS is defined on the NG nodegroup; so the data for table ORDERS will be spread across nodes 0, 1 and 2. To add node 1 to nodegroup NG, the utility will:

1. Add and redistribute data for table ORDERS on nodes 0 and 2 in nodegroup NG to node 1 which is added to nodegroup NG.
2. Insert entry for node 1 to the SYSIBM.SYSNODEGROUPDEF catalog table.
3. Generate a new partitioning map for nodegroup NG.

List the output of the nodegroups before adding node 1 by issuing:

```
db2 list nodegroups show detail
```

The output:

```
NAME                PMAP_ID  NODENUM IN_USER
------------------- -------  ------- -------
IBMCATGROUP               0        2  Y
IBMDEFAULTGROUP           1        0  Y
IBMDEFAULTGROUP           1        1  Y
IBMDEFAULTGROUP           1        2  Y
IBMDEFAULTGROUP           1        3  Y
IBMDEFAULTGROUP           1        4  Y
NG                        4        0  Y
NG                        4        2  Y
```

Connect to the database from the catalog node and issue:

```
db2 "redistribute nodegroup NG uniform add node(1)"
```

List the output of the nodegroups after adding node 1 by issuing:

```
db2 list nodegroups show detail
```

The output:

```
NAME                PMAP_ID  NODENUM IN_USER
------------------- -------  ------- -------
IBMCATGROUP               0        2  Y
IBMDEFAULTGROUP           1        0  Y
IBMDEFAULTGROUP           1        1  Y
IBMDEFAULTGROUP           1        2  Y
IBMDEFAULTGROUP           1        3  Y
IBMDEFAULTGROUP           1        4  Y
NG                        2        0  Y
NG                        2        1  Y
NG                        2        2  Y
```

This example shows how to use the redistribute nodegroup utility to add node 3 implicitly in the nodegroup, NG, which is defined on nodes 0, 1 and 2. The table, ORDERS, is defined on the nodegroup NG so that the data for table ORDERS will be spread across nodes 0-3. To add node 3 to NG implicitly, a target partitioning map which has nodes 0-3 specified is used. The utility will:

1. Add and redistribute data for table ORDERS on nodes 0-2 in nodegroup NG to node 3, which is added to nodegroup NG.
2. Insert an entry for node 3 to the SYSIBM.SYSNODEGROUPDEF catalog table.
3. Catalog the target partitioning map for nodegroup NG.

List the output of the nodegroups before adding node 3 by issuing:

```
db2 list nodegroups show detail
```

The output:

```
NAME                PMAP_ID  NODENUM IN_USER
------------------- -------  ------- -------
IBMCATGROUP               0        2  Y
IBMDEFAULTGROUP           1        0  Y
IBMDEFAULTGROUP           1        1  Y
IBMDEFAULTGROUP           1        2  Y
IBMDEFAULTGROUP           1        3  Y
IBMDEFAULTGROUP           1        4  Y
NG                        2        0  Y
NG                        2        1  Y
NG                        2        2  Y
```

Connect to the database from the catalog node and issue:

```
db2 "redistribute nodegroup NG using TARGETMAP $HOME/anodes.pmap"
```

List the output of the nodegroups after adding node 3 by issuing:

```
db2 list nodegroups show detail
```

The output:

```
NAME                PMAP_ID  NODENUM IN_USER
------------------- -------  ------- -------
IBMCATGROUP               0        2  Y
IBMDEFAULTGROUP           1        0  Y
IBMDEFAULTGROUP           1        1  Y
IBMDEFAULTGROUP           1        2  Y
IBMDEFAULTGROUP           1        3  Y
IBMDEFAULTGROUP           1        4  Y
NG                        3        0  Y
NG                        3        1  Y
NG                        3        2  Y
NG                        3        3  Y
```

5.7.9 Dropping Nodes from a Nodegroup

You can use the redistribute nodegroup utility to explicitly or implicitly drop existing nodes from a nodegroup. To drop nodes implicitly, you use a target partitioning map which does not contain the nodes to be dropped.

When you drop nodes from a nodegroup, the utility will:

1. Move the data on the nodes to be dropped to other nodes in the nodegroup.
2. Update the SYSIBM.SYSNODEGROUPDEF catalog table.
3. Generate a new partitioning map for the nodegroup if you drop nodes explicitly.
4. Catalog the target partitioning map as the new partitioning map for the nodegroup if you drop nodes implicitly.

5.7.9.1 Examples of Dropping Nodes from a Nodegroup

The following are examples of explicitly and implicitly dropping nodes from a nodegroup.

This example shows how to use the redistribute nodegroup utility to drop node 4 explicitly in the nodegroup, NG, which is created on nodes 0, 1, 2, 3, and 4. The table, ORDERS, is defined on the nodegroup, NG, so that the data for table ORDERS is spread across nodes 0-4. To drop node 4 from nodegroup NG, the utility will:

1. Remove the data for table on node 4, and redistribute the data across nodes 0-3.
2. Delete node 4 from the SYSIBM.SYSNODEGROUPDEF catalog table.
3. Generate a new partitioning map for nodegroup NG.

List the output of the nodegroups before dropping node 4 by issuing:

```
db2 list nodegroups show detail
```

The output:

```
NAME                PMAP_ID  NODENUM IN_USER
------------------- -------  ------- -------
IBMCATGROUP               0        2  Y
IBMDEFAULTGROUP           1        0  Y
IBMDEFAULTGROUP           1        1  Y
IBMDEFAULTGROUP           1        2  Y
IBMDEFAULTGROUP           1        3  Y
IBMDEFAULTGROUP           1        4  Y
NG                        3        0  Y
NG                        3        1  Y
NG                        3        2  Y
NG                        3        3  Y
NG                        3        4  Y
```

Connect to the database from the catalog node and issue:

```
db2 "redistribute nodegroup NG uniform drop node(4)"
```

List the output of the nodegroups after dropping node 4 by issuing:

```
db2 list nodegroups show detail
```

The output:

```
NAME                PMAP_ID  NODENUM IN_USER
------------------- -------  ------- -------
IBMCATGROUP               0        2  Y
IBMDEFAULTGROUP           1        0  Y
IBMDEFAULTGROUP           1        1  Y
IBMDEFAULTGROUP           1        2  Y
IBMDEFAULTGROUP           1        3  Y
IBMDEFAULTGROUP           1        4  Y
NG                        2        0  Y
NG                        2        1  Y
NG                        2        2  Y
NG                        2        3  Y
```

This example shows how to use redistribute nodegroup utility to drop nodes 1 and 3 implicitly in the nodegroup, NG, which is created on nodes 0, 1, 2, and 3. The table, ORDERS, is defined on the nodegroup, NG, so that the data for table ORDERS is spread across nodes 0-3. To drop nodes 1 and 3 from NG implicitly, a target partitioning map is used which has only nodes 0 and 2 specified. The utility will:

1. Move the data for the table on nodes 1 and 3 to nodes 0 and 2 according to the target map specified.
2. Delete nodes 1 and 3 from the SYSIBM.SYSNODEGROUPDEF catalog table.
3. Catalog the target partitioning map for nodegroup NG.

List the output of the nodegroups before dropping nodes 1 and 3 by issuing:

```
db2 list nodegroups show detail
```

The output:

```
NAME                PMAP_ID  NODENUM IN_USER
------------------- -------  ------- -------
IBMCATGROUP               0        2  Y
IBMDEFAULTGROUP           1        0  Y
IBMDEFAULTGROUP           1        1  Y
IBMDEFAULTGROUP           1        2  Y
IBMDEFAULTGROUP           1        3  Y
IBMDEFAULTGROUP           1        4  Y
NG                        3        0  Y
NG                        3        1  Y
NG                        3        2  Y
NG                        3        3  Y
```

Connect to the database from the catalog node and issue:

```
db2 "redistribute nodegroup NG using TARGETMAP /$HOME/rnodes.pmap"
```

List the output of the nodegroups after dropping nodes 1 and 3 by issuing:

```
db2 list nodegroups show detail
```

The output:

```
NAME                PMAP_ID  NODENUM IN_USER
------------------- -------  ------- -------
IBMCATGROUP               0        2  Y
IBMDEFAULTGROUP           1        0  Y
IBMDEFAULTGROUP           1        1  Y
IBMDEFAULTGROUP           1        2  Y
IBMDEFAULTGROUP           1        3  Y
IBMDEFAULTGROUP           1        4  Y
NG                        4        0  Y
NG                        4        2  Y
```

5.7.10 Failure Recovery

The data redistribution operation might fail at an intermediate point. This may be caused by lack of space for logs and data files or by network/system problems. Data redistribution is performed one table at a time, and if a failure occurs, some tables may have been redistributed while others were not. There are two recovery methods to take if a failure occurs:

Continue option Continue to redistribute all remaining tables. If the cause of the problem has been found and fixed, this may be the easiest method to complete the operation.

Rollback option Undo the redistribution, and revert redistributed tables back to their original state.

During the execution of the `redistribute nodegroup` operation, a message file is written to the $HOME/sqllib/redist directory. The filename has the following format:

database-name.nodegroup-name.timestamp

Note: The timestamp value is the time when the command is issued.

5.7.10.1 Example of a Failed Redistribution

The example given indicates the failed redistribution process and the steps taken after the failure.

The following message file was produced when a user, using the redistribute nodegroup command, tried to add nodes 4 and 5 to the nodegroup, NG, that resided in database DSS. The command issued was:

```
db2 "redistribute nodegroup NG uniform add node(4 to 5)"
```

The message file DSS.NG.19951022133222:

```
                    Data Redistribution Utility
                    ___________________________

 The following options have been specified:
 Nodegroup name                 : NG
 Operator                       : U          1
 Redistribute Nodegroup         : uniformly
 No. of nodes to be added       : 2
 List of nodes to be added      :            2
  4
  5
 No. of nodes to be deleted     : 0
 List of nodes to be deleted    :

 The execution of the Data Redistribution operation on:

  Table                                  begun at  ended at
  _____                                  ________  ________
DB2PUSER.ORDERS                          13.32.25            3

 --Data Redistribution cannot be continued.--

 Error: Redistribution failed with SQLCODE=-1031 (save_rc=-2200). 4
```

The description of the output is as follows:

1 Indicates the data redistribution in this example was done to achieve a balanced distribution. This shows the type of data redistribution to be done. Possible values are:

U To uniformly redistribute the nodegroup to achieve a balanced distribution.

T To redistribute the nodegroup using targetmap.

C To continue a redistribution operation that failed.

R To roll back a redistribution operation that failed.

2 Indicates that the list of nodes to be added in this example are nodes 4 and 5.

3 Indicates the list of tables in the nodegroup.

4 Indicates the redistributed operation failed with SQLCODE -1031. The error code means:

```
The database directory cannot be found on the
indicated file system.
```

The failure occurred because the database partition was not created on node 5. To resolve this failure, the following steps can be taken:

1. Log onto the host for node 5.
2. Set the DB2NODE environment variable to 5 if using a logical node
3. Issue the add node command to create the database partition on node 5.

After the add node command completes successfully, issue the following command to continue:

```
db2 "redistribute nodegroup NG continue"
```

The following message file is generated when the previous data redistribution continues after the failure.

```
                    Data Redistribution Utility
                    ___________________________

 The following options have been specified:
 Nodegroup name                 : NG
 Operator                       : C
 No. of nodes to be added       : 0
 List of nodes to be added      :
 No. of nodes to be deleted     : 0
 List of nodes to be deleted    :

 The execution of the Data Redistribution operation on:

  Table                                  begun at  ended at
  _____                                  ________  ________
DB2PUSER.ORDERS                          15.08.42
                                                   15.08.55

 --All tables in the nodegroup have been successfully redistributed.--
```

This message file indicates the continuation of the data redistribution that failed has been successfully completed.

5.8 Runstats Utility

The runstats utility is used to collect the statistics of the table and its associated indices. The statistics consist of information such as the number of records, the number of pages and the average record length.

5.8.1 Using Runstats

The syntax of the Runstats utility and the authorization to use the utility in DB2 Parallel Edition are the same as they are in DB2/6000 Version 1. For more information, see the *DB2/6000 Command Reference*.

In DB2 Parallel Edition, the runstats operation:

- Collects statistics about the table and its indicies by executing the RUNSTATS utility at a single node. The node at which the RUNSTATS utility is determined by the following:
  - If the node where the RUNSTATS is issued contains a partition for the table, the utility executes at this node.
  - If the node where the RUNSTATS is issued does not contain a table partition, the utility sends the request to the first node in the nodegroup that holds the table partition. The statistics related to the table and its indicies will be collected at that node.
- Derives the global table and index statistics by multiplying the collected statistics by the number of nodes over which the table is partitioned.
- Stores the global statistics in the system catalog table.

The parallel optimizer uses these statistics to determine the optimal access path to the data.

The following items need to be considered when using the runstats utility:

- You must be connected to the database to execute this command.
- To create new access paths to the table after its statistics are updated, make sure to rebind the packages.
- If the RUNSTATS operation is run on a node that is not a member of the nodegroup in which the table is created, the request will be sent to the first node of the nodegroup that holds the table partition. The RUNSTATS utility then executes at that node.
- The DB2 Parallel Edition SQL optimizer assumes that data are uniformly distributed across the nodes of the system. If the distribution is not uniform, it is recommended that you run the RUNSTATS utility on a node that has a representative data distribution.

The following example collects statistics for both the table (ORDERS) and its indexes:

```
db2 CONNECT TO dss
db2 RUNSTATS ON TABLE  db2puser.orders AND INDEXES ALL
db2 CONNECT RESET
```

5.9 Reorgchk Utility

The reorgchk utility runs statistics on the tables in a database to determine if they need to be reorganized.

5.9.1 Using Reorgchk

The syntax of the reorgchk utility and the authorization required to use the utility in DB2 Parallel Edition are the same as they are in DB2/6000 Version 1. For more information, see the *IBM DATABASE2 Parallel Edition for AIX Administration Guide and Reference*.

The reorgchk operation:

- Calls runstats to gather statistics on the table you specify.
- Uses the statistics to calculate the six formulas to determine if reorganization is required.
- Generates the report.

The following are considerations when using the Reorgchk utility:

- Each (*) under the REORG column of the report indicates the results of the calculations exceed the bounds set by the formula, and table reorganization is suggested.

The following example runs statistics on table PORDER to determine if it needs to be reorganized.

```
db2 CONNECT TO dss
db2 REORGCHK UPDATE STATISTICS ON TABLE db2puser.porder
db2 CONNECT RESET
```

The following output is generated from the previous reorgchk command.

```
Doing RUNSTATS ....


Table statistics:

F1: 100*OVERFLOW/CARD < 5
F2: 100*TSIZE / ((FPAGES-1) * 4020) > 70
F3: 100*NPAGES/FPAGES > 80

CREATOR  NAME                 CARD    OV    NP    FP    TSIZE  F1  F2 F3 REORG
-------------------------------------------------------------------------------
DB2PUSER PORDER               1098     0    24    24    83448   0  90 100 ---
-------------------------------------------------------------------------------

Index statistics:

F4: CLUSTERRATIO > 80
F5: 100*(KEYS*(ISIZE+10)+(CARD-KEYS)*4) / (NLEAF*4096) > 50
F6: 90*(4000/(ISIZE+10)**(NLEVELS-2))*4096/ (KEYS*(ISIZE+10)+(CARD-KEYS)*4)<100

CREATOR  NAME                CARD  LEAF  LVLS ISIZE   KEYS   F4   F5  F6 REORG
-------------------------------------------------------------------------------
Table: DB2PUSER.PORDER
DB2PUSER DATE_INDX           1098     3     1     4     24  100   37    - -*-
SYSIBM   SQL951113152800640  1098     6     2    10   1098  100   89   16 ---
-------------------------------------------------------------------------------

CLUSTERRATIO (F4) will indicate REORG is necessary for indexes that are not in
the same sequence as the base table. When multiple indexes are defined on a
table, one or more indexes may be flagged as needing REORG.  Specify the most
important index for REORG sequencing.
```

5.10 Reorganize Table Utility

The Reorganize Table utility reorganizes a specified table and it's associated indices by reconstructing the rows to eliminate fragmented data as in serial DB2. The only difference for DB2 Parallel Edition for this operation is that the reorganization of the table executes in parallel at each node of the nodegroup in which the table is defined. Even if the reorganize operation fails on one or more nodes, the rest of nodes, which successfully completed, will not be rolled back.

An index on a table under DB2 PE is made of the local indices in that table on each node in the nodegroup.

5.10.1 Using Reorganize Table

The syntax of the Reorganize Table utility and the authorization to use the utility in DB2 Parallel Edition are the same as they are in DB2/6000 Version 1. For more information, see the *IBM DATABASE2 Parallel Edition for AIX Administration Guide and Reference*.

The Reorganize Table utility operates at each node in the following way:

1. Create a temporary table.
2. Open a cursor on the reorganized table partition with one of the following SQL statements:

- SELECT * FROM creator.tablename (if no index is specified in the reorg table).
- SELECT * FROM creator.tablename ORDER BY colname1, ..., colnameN
  (if index is specified in the reorg table command; the columns in the ORDER BY are the columns that make up the index.)

3. Fetch a row using the cursor.
4. Insert the row into the table.
5. Replace the original table with the reorganized table.
6. Recreate all the indices (if any exist).

The following items need to be considered when using the Reorganize Table utility:

- You must be connected to the database to execute the command.
- Ensure there is enough space to do the reorganization.
- For a table that is larger than 2 GB, you must ensure the file systems are mounted at the directories where the temporary tables are created before proceeding with the table reorganization.
- After successfully reorganizing a table, you should update the table statistics using RUNSTATS and then rebind the packages that use the reorganized table.

5.11 Backup and Restore

The concepts of backup and restore on a parallel database system are similar to those on a serial database system. Some of the concerns that are different are the following:

- Managing many nodes, each of which may hold gigabytes of data
- Allowing for sufficient resources to perform backup and restore on large databases
- Planning a backup strategy that will be able to provide a recovery plan in the shortest amount of time
- Keeping track of, and allowing for, the storage of backups

This section will give only an overview and highlight some of the differences in a parallel database environment. More information can be found in the *DB2 Parallel Edition for AIX Administration Guide and Reference*.

The DB2 PE backup/restore utility has the same syntax that it has in serial DB2/6000. The backup and restore commands will only operate on a single-node basis. To backup a complete database, the command must be run on every node over which the database is spread. Restore and roll forward are done on a node-by-node basis. All nodes do not have to be backed up at the same time.

5.11.1 Backup and Restore Scenario

Figure 64 shows a backup and restore scenario in DB2 PE.

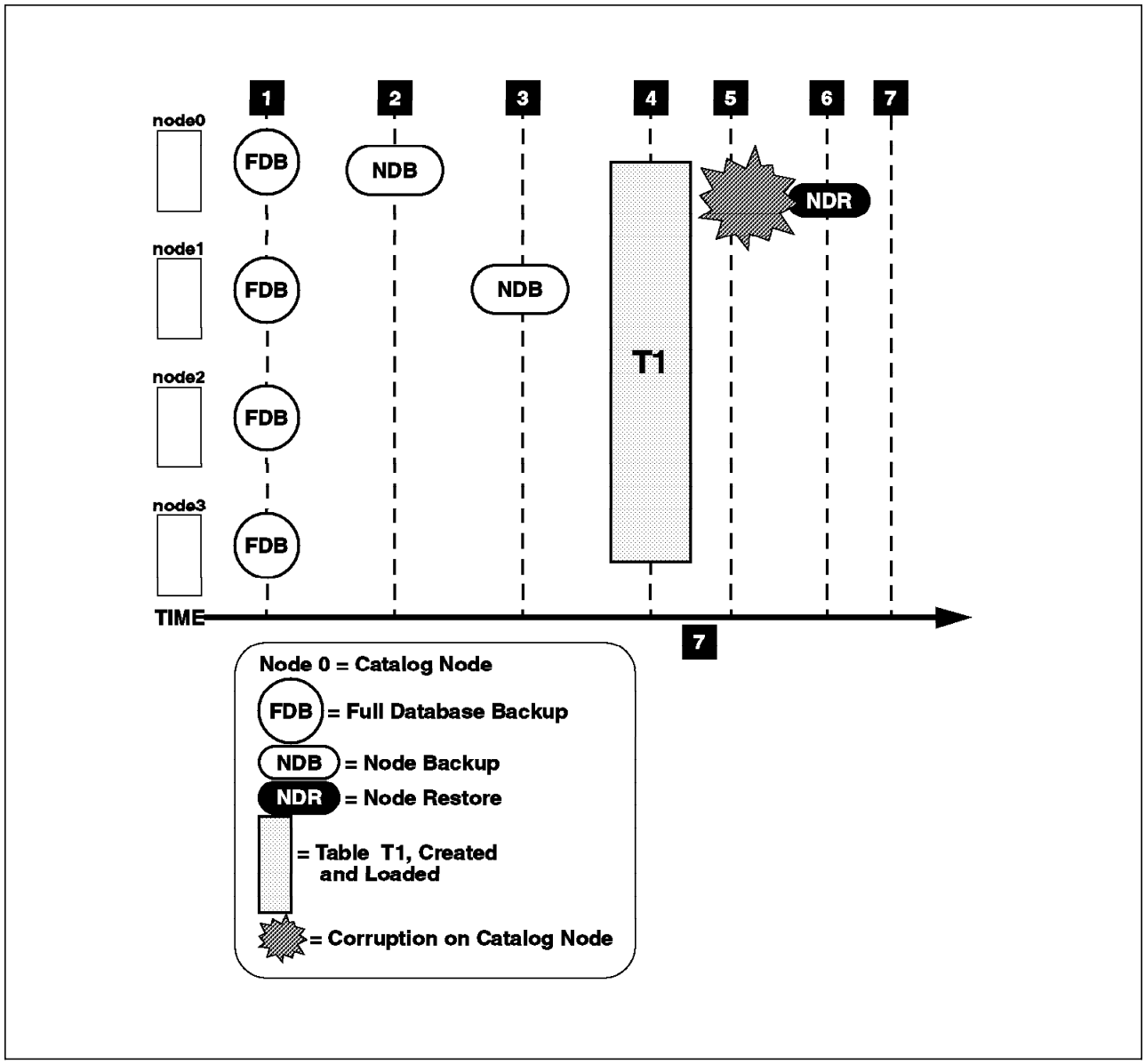


Figure 64. Backup, Restore and Roll-Forward Operations

Figure 64 can be explained as follows:

1 The database was changed from the default of circular logging to log retain (archived logging). A full database backup must be performed on all nodes for the change to take effect.

2 Although incremental backup is not supported in DB2 PE, backup can be done at the node level. You need not back up all the nodes at once. Depending on your environment, you could back up one node every day. In our 4-node example, this would give you a 4-day rotation period. At any point in time, the maximum recovery for a node is the restore of the backup and applying 3 days worth of log activity.

You can do backup and restore of different nodes in parallel. The one restriction is that you cannot do concurrent offline backup or restore of the catalog node with non-catalog nodes. In our example, node 0 is the catalog node. The catalog node could also be on a logical node with no user data. This will shorten the time in recovery.

3 This is another node backup. DB2 Parallel Edition supports multiple logical nodes as well as physical nodes. This means that you can partition data at a finer level than the number of actual physical nodes. Suppose you have 20 GB of data on each physical node. You could partition two logical nodes for each physical node (10 GB per logical node). This will help in managing smaller backup elements.

4 A table was loaded using the load utility provided by DB2 PE. Note that DB2 PE does not perform logging during the load. There is a log record that indicates that a table in the database was populated by the Load utility. If the Rollforward utility encounters such a record, the table will be marked as unavailable. The table will be dropped by the Rollforward utility if it later encounters a DROP TABLE log record. Otherwise, after the database is recovered, SQL1477N will be issued if any attempt is made to access the table. You should drop the table. If you want to use the table again, re-create it.

5 An incorrect application was executed, and its changes were committed in the database. This caused a corruption of the database.

6 We will restore the database from a backup and do a roll-forward recovery to a point in time prior to, but not including, the application that caused the corruption.

7 This is the point in time to which we will do a roll-forward. However, this point in time backup does not include the load operation of the table, T1. The load utility can perform as fast as a backup and restore. So, recovery can be performed just as quickly by reloading. The table should be dropped and recreated and then reloaded. The other consideration is that the source file for the load must be available for the load operation.

5.11.2 Backup Operation

There are two kinds of backup methods provided, offline backup and online backup. While an offline backup is running on a node, the database cannot be used at the node. During an online backup, the database can be accessed, and data in the table can be updated. To do this, the logging mode of the database must be set to “retained.”

DB2 PE provides a new mode, called exclusive at node, for connecting to a database. It prevents other applications from making a connection to the database on the node, but allows them to connect to other nodes. DB2 PE uses this new mode in connecting for the offline backup and restore of the database. For online backup, DB2 PE uses shared mode.

The backup file name format has changed from serial DB2/6000 by adding the node number. For example:

```
DSS.db2pe.N4C0.19951107101429.001
```

This is generated using the parts shown below:

Database alias The database alias in our example is DSS.

PE instance name The instance owner is db2pe.

Node number This example indicates that the database was created on node 0, and backed up on node 4.

Timestamp The timestamp is divided into year, month, day, hour, and minute.

Sequence number This is a three-digit number that is used as a file extension.

The `list nodes` command can be used to show the list of nodes that are contained in a nodegroup of a database. An example of this is shown below:

```
db2 list nodes

  NODENUM              NODENAME
---------------      ---------------
    0                  NODE000000
    1                  NODE000001
    2                  NODE000002
    3                  NODE000003
```

Backups will need to be run on all the nodes listed by this command in order to have a full backup.

A utility script, db2_all, is provided which will submit backup/restore commands at multiple nodes. This is held in $HOME/sqllib/misc. This script is used to execute the same database command on one or more nodes at the same time. The following is an example of the use of the command to back up database dss on all nodes:

```
db2_all "db2 backup database dss to /dbbackup"

DSS.db2pe.N0C0.19951107102105.001
DSS.db2pe.N1C0.19951107103006.001
DSS.db2pe.N2C0.19951107102938.001
DSS.db2pe.N3C0.19951107102431.001
```

The backup image of the data on each node has changed slightly from serial DB2. It now contains:

- The node number to distinguish it from backup images of other nodes of the same database.
- The catalog node number of the database.
- The release number (different from serial DB2 release number). This prevents a backup of a serial database from being restored into DB2 PE.

DB2 PE has three kinds of media interfaces for storing the database backup image as does serial DB2/6000. The valid types are disk, tape or ADSM.

Since AIX files are currently limited in size, if the database is backed up to disk, the size of the database on each node must not exceed 2 GB.

ADSTAR Distributed Storage Manager (ADSM) is a product used to manage file or database backups centrally. The ADSM server supports many storage devices on many different platforms. DB2 PE uses ADSM API V1.2.0 to communicate with the ADSM server for backing up the database.

5.11.3 Restore

The `restore` command is used to rebuild a damaged or corrupt local database at each node by using the log and the backup image created by the `backup database` command. It returns the database on the node to the state of the database when the backup was formed for the node. If the backup copy was created during an online backup, forward recovery is invoked automatically at the end of restore operation using the logs. This concept is the same as it is in serial DB2/6000. The only difference is that DB2 PE restores the database portion separately at each node using local logs, while serial DB2/6000 restores the whole database with a single log.

The restore utility will create a new database if the target database does not exist. If the `restore` command (to new database) is issued on multiple nodes at the same time, all of the `restore` commands will try to create the same database. As a result, the `restore` command will fail. Restore on the catalog node first to avoid this problem. After a new database is created by this operation, the restore operation at other nodes can be performed.

Because the restore utility makes a connection to the database, if the database portion is damaged at a node, the connection cannot be made. In this case, the restore utility fails with SQL2010N. The `drop database at node` and `create database at node` commands should be used in this case to recreate the database portion on the failed node.

5.11.4 Restrictions

There are some restrictions for DB2 PE backup:

- Roll-forward recovery can only be invoked from the catalog node. If the catalog node requires forward recovery, the roll-forward utility will hold an exclusive connection to this node until the recovery has successfully completed. The database is not available for use for any other application until this completion has occurred. If the catalog node does not need to be rolled forward, the utility will use a shared connection to perform the roll forward. Nodes that are not being rolled forward are then available for use by other applications.
- Concurrent offline backup of both catalog and non-catalog nodes is not possible. As the offline backup of the catalog node makes

an exclusive connection to the catalog node, no one can access the catalog node during the backup. For this reason, you might want to consider placing the catalog node on a logical node with no user data. This will help to speed recovery time.

- Create/Drop/Alter Table is not allowed on any node while a backup is executing on a node in online mode.
- Since a backup image includes the node number, it can only be restored to a node with same node number. It is not possible to restore a backup from another logical node. DB2 PE makes use of hash distribution. The data on node A is hashed to that node and cannot be moved to node B by a backup or restore operation.
- Since a backup image includes a catalog node number, restoring to an existing database with a different catalog number is not allowed.
- Restoring to a new database must be done first at the catalog node. Otherwise, the SQLCODE -6026 will be returned indicating on which node the restore command must be issued first.

5.12 Recovery

In the DB2 PE environment, a database is spread across multiple nodes, and each node does its own logging and backup. Any transaction running under DB2 PE must be coordinated across the nodes. For these reasons, detection of inconsistency between nodes, coordinated forward recovery of the database, maintenance of consistency for each transaction, and the sequence of transactions across nodes become important issues.

To preserve consistency of the database between the nodes, the first operation during a database connection on each node is to check log file ID and log record number that are stored in the master log control file on the catalog node.

5.12.1 Database Logs

The database log is an important resource used in recovery. The database log at each node has a unique log path and uses the node name in the path.

The default log path is the SQLOGDIR subdirectory in the database directory at every node. For example, on node 1, this may be:

```
/$DB2INSTANCE/NODE00001/SQL00001/SQLOGDIR
```

When the newlogpath configuration parameter is changed, the node name will be automatically appended to the end of the specified path.

When disks are connected to a single processor, loss of that processor means loss of access to those disks. Without the log held on those disks, it is impossible to restore from backup and roll-forward changes. To protect against this, multipath disks are strongly recommended for the log.

5.12.2 Virtual Timestamps

Under serial DB2/6000, when a transaction is committed, the system clock on the machine is used as the transaction time stamp. This guarantees the commit sequence of transactions. DB2 PE cannot use this method because each transaction is run across on multiple nodes which may have different local system clocks.

To avoid database inconsistency across the nodes, DB2 PE use a concept known as virtual time. Each node in the database keeps a virtual clock. It can only be adjusted forward. It is always kept ahead of the node′s local system clock. When a transaction is committed, it uses the most advanced virtual clock over all the nodes that are involved in the commit operation. A transaction′s time stamp is defined as its commit time and is called its virtual timestamp (VTS). The VTS will be stored in the log and at the same time, all other nodes involved in that transaction will have their virtual clocks adjusted to the chosen VTS.

Figure 65 illustrates this process. Let us suppose there are four update SQL statements involved in the transaction. Each of the update SQL statements will be executed at host0, host1, host2, and host3 in parallel. The four nodes have different virtual clock times at the transaction′s commit time. The most advanced virtual clock is set to four o′clock on host 3. It is used as the virtual timestamp and stored in the log at each node. At the same time, the virtual clock at each node, except host3, will be adjusted to 4 o′clock.

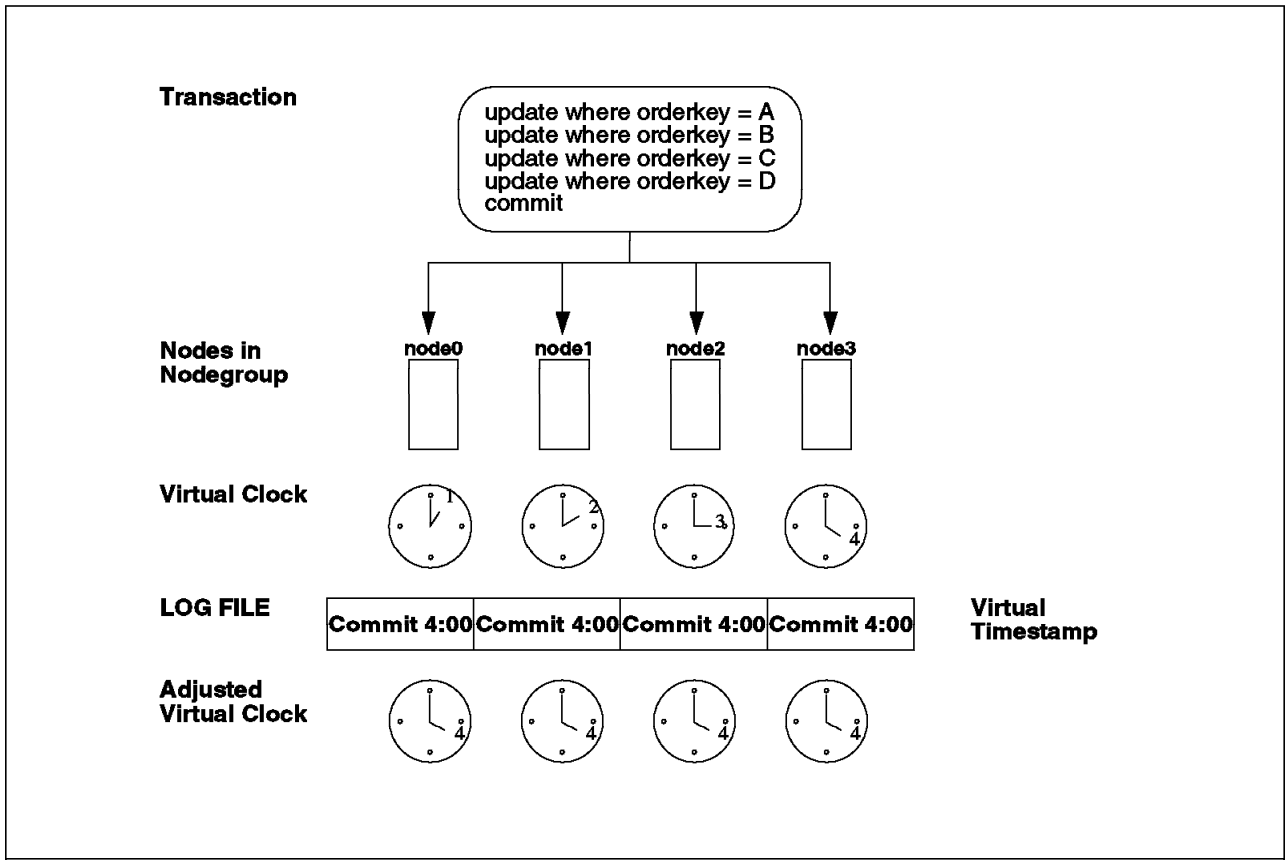


Figure 65. Virtual Time Stamp

When two different transactions update or reference the same row of the same table on a node, they are said to be related. Using the virtual time method, the most advanced clock on all the nodes involved in a transaction is used as the VTS. Using this method DB2 PE can guarantee the commit sequence for related transactions and can use the timestamps during database recovery.

There is a database manager configuration parameter, max_time_diff, that can help reduce the possibility of a node having a system clock that could push ahead the time for all the nodes defined in a system. This parameter holds a value that is used as the maximum amount of time in which two system clocks can differ. The following relate to the max_time_diff parameter:

- The possible values for this parameter are from one minute to 24 hours. The default is one hour.
- When one node first attempts to connect to the database on another node, the catalog node for the database checks to see that the time on the node requesting the connection and the time on the node that is to be connected to is within the limit specified in the max_time_diff parameter. If the value specified in the max_time_diff parameter is exceeded, the connection is not allowed.
- An update transaction that involves more than two nodes in the database must verify that the time on the participating nodes is synchronized before the update can be committed. If two or more nodes have a greater time difference than the value that is stored in max_time_diff, the transaction is rolled back to stop the incorrect time from being distributed to the other nodes.

5.12.3 Point-In-Time Recovery

When you do point-in-time recovery, you rollforward changes to a specific point in time. This means that you need a backup image from before this time, and the logs from the time of the backup until the point in time you want to recover to.

Point-in-time recovery is done at the database system level; that is, on all the node listed in the db2nodes.cfg file, even those that do not contain user data. Before you can do point-in-time recovery, you must restore the database on all nodes and ensure that each database partition is set to the rollforward-pending state.

5.13 Governor Utility

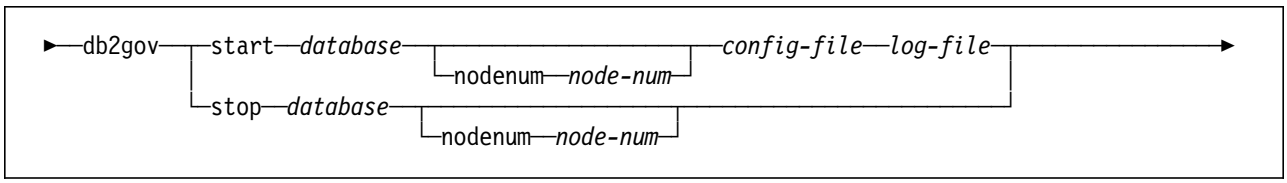
DB2 Parallel Edition V1.2 provides a utility called the governor. The governor utility is designed to work on multiple nodes in parallel. It is used to monitor and change the behavior of applications that run against a database according to the rules specified by user in the governor configuration file. It consists of the following parts:

- The governor front-end utility
- The governor daemon
- The governor configuration file
- The governor log files
- The governor log query utility

For more information about the governor utility, refer to *DB2 Parallel Edition for AIX Administration Guide and Reference*.

5.13.1 Governor Front-End Utility

The governor front-end utility, db2gov, is used to start and stop the governor daemon (on either all logical nodes or on a single node). You must have SYSADM authority to use the utility. The syntax is as follows:



where the parameters are as follows:

database Use to specify either the database name or the database alias. Make sure the specified database has the same name as that specified in the governor configuration file.

node-num Use to specify the node number where the governor daemon should be started. Make sure the specified node number is the same as that specified in the $HOME/sqllib/db2nodes.cfg.

config-file Use to specify the name of the configuration file where user has defined rules for the governor daemon to use for monitored resource and took actions. The config-file can reside in one of the following directories:

- The default directory $HOME/sqllib
- The directory of the fully specified path name
- The current directory

log-file Use to specify the base name of the log files the governor is to write to. The log files are stored in the $HOME/sqllib/log directory. The node number on which the governor daemon is running is automatically appended to the log files.

When starting the governor daemon with the db2gov utility, the db2_all processes spawned will not exit until the governor daemons are stopped. To have db2gov start the governor daemons without leaving the db2_all processes around, add the *""* flag to the db2_all commands in the db2gov shell script:

```
db2_all ';"' db2gov start $Database nodenum '##' $ConfigFile $LogFile
db2_all ';"' db2gov stop $Database nodenum '##'
```

5.13.2 The Governor Daemon

The governor daemon is a process that executes "in the background" to monitor and change the behavior of applications that run against a database based on the set of rules defined in the governor configuration file.

The following describes the steps involved in the governor daemon after it is started:

1. Call the sqlmon (Database System Monitor Switch) API to active the unit of work switch.
2. Check to see if the governor configuration file has changed or has not yet been read. If either condition is true, it reads the rules in the governor configuration file.
3. Issue the snapshot request to obtain statistics about the applications running against the database which it is monitoring.
4. Check the obtained statistics against the rules specified in the governor configuration file and take action accordingly. The action can be:
   - Force the application
   - Change the priority of all the agents working on behalf of the application on that node.
   - Write a record of any action it takes to the governor log file.
   - Sleep until the interval specified in the governor configuration file is exceeded. If no interval is specified, the default interval of 120 seconds is used.

The governor daemon only exits if it encounters an error or it is stopped by the governor front-end utility. Before exiting, the governor daemon will do the following:

- Obtain a snapshot of all the agents running on the database
- Reset the priority of the agents

5.13.3 Customizing the Governor Configuration File

The governor configuration file is used to define and configure the rules which the governor uses to govern applications running against the database. It can be changed without stopping the governor.

The governor configuration file must be created in a directory that is accessible by the governor daemon on each node. You can specify comment in the file by delimiting the text within the { } braces.

The rules in the configuration file consist of:

- The name of the database or its alias to which the rules apply. This rule is only specified once in the file.

dbname database-name

- The interval the governor sleeps before waking up to check the behavior of the applications. This rule is only specified once in the file.

  interval the interval in seconds (Default of 120 seconds)

- The rule that specifies how to govern the applications. The rule clauses can be combined to form a rule. The clauses can only be specified once in a rule but can be specified in more than one rule. Each rule in the file must be followed by a semicolon (;). The following shows the description of the rule clauses and it must be specified in the order shown. Square brackets ([ ]) indicate an optional clause.

  – The text description of the rule. It must be enclosed by either single or double quotation marks.

    [desc] 'description'
    or
    [desc] "description"

  – The time period during which the rule is to be applied. If this clause is not specified, the rule is valid 24 hours a day.

    [time] hh:mm hh:mm

  – The rule applies to one or more authorization ids under which the application is executing. Multiple authids must be separated by a comma (,). The default is that the rule will apply to all authorization ids.

    [authid] userid1, userid2,...

  – The rule applies to the name of the executable that makes the connection to the database. Multiple application names must be separated by a comma (,). The default is the rule will apply to all application names.

    [applname] appl1, appl2,...

    Note: Application names are case-sensitive and can only have 20 characters. Otherwise, it will be truncated to 20 characters.

  – The rule applies to one or more resource limits for the governor to check. The limits can only be -1 or greater than 0.

    [setlimit] limit

    The governor can check the following limits and at least one of the limits must be specified:

    - The limit of the number of CPU seconds that can be consumed by an application. If -1 is specified, the governor does not limit the application's CPU usage.

      cpu nnn

    - The limit of the number of locks that an application can hold. If -1 is specified, the governor does not limit the number of locks held by the application.

      locks nnn

- The limit of the number of rows that an application can select. If -1 is specified, the governor does not limit the number of rows that can be selected.

  rowssel nnn

- The limit of the number of seconds for the elapsed time of a unit of work. If -1 is specified, the elapsed time is not limited.

  uowtime nnn

– The rule to specify the action to take if one or more of the specified limits is exceeded. The action can be one of the following:

  - Change the priority of the agents working for the application. (The priority of the agents is set by the AIX nice command). The valid values are from -20 to 20., where a lower value assigns a higher priority to the agents. Negative nice values should be used with caution. They give the database agents priority over all other user processes, even the database engine itself!

    [action] priority nnn

  - Force the agent that is servicing the application.

    [action] force

    Note: If a limit is exceeded and the action clause is not specified, the governor reduces the priority of the agents working for the application by 10.

Note: If more than one rule is specified in the file, the last applicable rule wins except if -1 is specified for a clause in a rule. In this case, the value specified for the clause in the subsequent rule can only override the value previously specified for the same clause. Other clauses in the previous rule are still operative. Each rule in the file must be followed by a semicolon (;).

5.13.4 Governor Log Files

To provide an audit trail of what the governor has done, records for the following actions done by the governor front end utility or the governor daemon and situations encountered by the governor daemon will be logged:

- Force an application
- Read the governor configuration file
- Change the agent's priority
- Encounter an error or warning
- Start the governor
- End the governor

A separate log file exists for each governor daemon. The log files are stored in the $HOME/sqllib/log directory. The node number of the node that the governor is running on is automatically appended to the log file name. The log file name is the log name used when

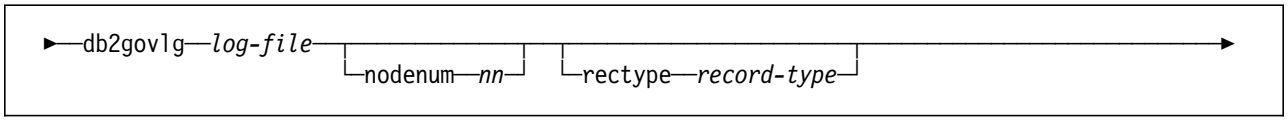
issuing the governor front end utility db2gov. Each log record has the following format:

```
Date Time NodeNum  Rectype  Message
```

- The format of the date field: YYYY-MM-DD
- The format of the time field: HH.MM.SS
- The nodenum field indicates the node number of the node on which the governor is running
- The Rectype field consists of the following possible values:
  - ERROR to indicate an error was encountered
  - FORCE to indicate an application was forced
  - NICE to indicate the priority of an application was changed
  - READCFG to indicate the configuration file was read by the governor
  - START to indicate the governor was started
  - STOP to indicate the governor was stopped
  - WARNING to indicate a warning was issued
- The message field provides more detailed descriptions for the value of the rectype field.

5.13.5 Governor Log Query Utility

The governor log query utility, db2govlg is used to query the log file. You can query the log files for a single node or for all nodes. The output will be sorted by date and time. You can also query the log based on the Rectype field. The syntax is as follows:



where the parameters are as follows:

log-file The base name of the log file(s) that you want to query. monitoring.

nn The node number of the node on which the governor is running.

record-type The type of record that you want to query. The possible types are:

- ERROR
- FORCE
- NICE
- READCFG
- START
- STOP
- WARNING

Note: There are no authorization restrictions for using the governor query log utility. If you want to restrict access to this utility, you can change the group permissions for the db2govlg file.

5.13.6 Considerations for the Governor Utility

The following is the list of the considerations when using the governor.

1. The governor daemon must be owned by root.
2. You require SYSADM authority to use the utility.
3. The root user must be a member of the SYSADM group (the group that owns the $HOME/sqllib directory) on all nodes on which the governor is to be run.
4. The agentpri database manager parameter must be set to the default value. Otherwise, it will override the priority clause set in the rule.
5. The governor requests snapshots of the database manager. It may affect CPU usage if the wake-up interval for the governor daemon is set too small. Consider increasing the wake-up interval if the governor daemon uses too much CPU.
6. When the governor daemon changes the priority of the agents working on behalf of an application, it also has to reset the priority of these agents if they are reused by a different application. There may be a slight chance that another application could use these agents during the interval between the time that an agent stops working for an application and the governor daemon resets the priority of that agent. This could affect the performance of the application because the agent is running at a nonstandard priority.

5.13.7 Examples Using Governor Utility

The following will serve as examples for this section.

- EXAMPLE 1 illustrates an error that has occurred when starting up the governor on a 4-node system.

  The following shows the steps to configure the governor configuration file and to start up the governor for the database dss on 4 nodes using the governor front-end utility.

  The configuration file for EXAMPLE 1 instructs the governor to check every 5 seconds for any application that selects over 1000 rows. An application that exceeds that limit will have the priority of its agents reduced by a default of 10.

  1. Customize the governor configuration file gov.cfg. The following shows its content:

     ```
     interval 5; dbname dss;
     setlimit rowssel 1000;
     ```

  2. Start up the governor using the following command:

```
db2gov start dss gov.cfg dssgov.log
```

The output from the screen after executing the db2gov command:

```
rah: db2pe
rah: db2pe
rah: db2pe
rah: db2pe

db2pe: db2gov: Starting db2govd for database dss on node 0
db2pe: db2gov start dss nodenum ... completed ok

db2pe: db2gov: Starting db2govd for database dss on node 1
db2pe: db2gov start dss nodenum ... completed ok

db2pe: db2gov: Starting db2govd for database dss on node 2
db2pe: db2gov start dss nodenum ... completed ok

db2pe: db2gov: Starting db2govd for database dss on node 3
db2pe: db2gov start dss nodenum ... completed ok
```

3. After running db2gov, perform the following steps:

   a. Check to make sure the governor daemon is up and running on all the nodes by issuing the following command:

      ```
      rah "ps -ef | grep db2govd | grep -v grep"
      ```

      If the db2govd process is not there, the governor daemon has not started successfully.

   b. Check the governor log files using the query log utility by executing the following:

      ```
      db2govlg  dssgov.log
      ```

      The following messages were produced in the output:

      ```
      1996-09-08 14.32.11 0 ERROR (15020) SQLMONSZ Error:
                                          SQLCode = -1092
      1996-09-08 14.32.11 0 ERROR (15020) SQLMONSZ Error:
                                          SQLCode = -1092
      1996-09-08 14.32.11 0 READCFG Config = /home/db2pe/gov.cfg
      1996-09-08 14.32.11 0 START   Database = DSS

      [ Messages for node 1 - 3 omitted - they are identical
      in detail ]
      ```

The -1092 SQLCODE indicates the user does not have the authority to perform the requested command. Correct the

problem by adding the root user to the SYSADM group for the instance using smitty.

- EXAMPLE 2 shows how to use the governor to monitor two database manager nodes and force any application that has exceeded the limit of 21,600 seconds for the elapsed unit of work time.

  1. Customize the governor configuration file gov.cfg. The following shows its content:

     ```
     interval 5; dbname dss;
     setlimit uowtime 21600 action force;
     ```

  2. Start up the governor using the following command:

     ```
     db2gov start dss2 gov.cfg gov_dss2.log
     ```

     The output from the screen after executing the db2gov command:

     ```
     rah: db2pe
     rah: db2pe

     db2pe: db2gov: Starting db2govd for database dss2 on node 0
     db2pe: db2gov start dss2 nodenum ... completed ok

     db2pe: db2gov: Starting db2govd for database dss2 on node 1
     db2pe: db2gov start dss2 nodenum ... completed ok
     ```

  3. After running db2gov, perform the following steps:

     a. Check to make sure the governor daemon is up and running on all the nodes by issuing the following command:

        ```
        rah "ps -ef | grep db2govd | grep -v grep"
        ```

        If the db2govd process is not there, the governor daemon has not started successfully.

     b. Check the governor log files using the query log utility by executing the following:

        ```
        db2govlg  gov_dss2.log
        ```

        The following messages were produced in the output:

        ```
        1996-09-10 19.20.05  0 READCFG  Config = /home/db2pe/gov.cfg
        1996-09-10 19.20.05  0 START    Database = DSS2
        1996-09-10 19.20.05  1 READCFG  Config = /home/db2pe/gov.cfg
        1996-09-10 19.20.05  1 START    Database = DSS2
        ```

If any application exceeds the unit of work limitation rule, it will be forced and the application will receive the following message:

```
SQL1224N  A database agent could not be started to service a
request, or was terminated as a result of a database system
shutdown or a force command
```

4. Check the governor log file for node 0 using the query log utility by executing the following:

```
db2govlg  gov_dss2.log nodenum 0
```

The following messages were produced in the output:

```
1996-09-10 19.20.05  0 START    Database = DSS2
1996-09-10 19.20.05  0 READCFG  Config = /home/db2pe/gov.cfg
1996-09-10 19.24.01  0 FORCE    applname db2bp
authid DB2PE applid *LOCAL.DB2.960910232031
coord 0 (line 1)  ET 21601
```

The entry in the log file indicates the governor has forced db2bp application.

- EXAMPLE 3 shows how to use the governor to favor short running queries over longer running queries:

1. Customize the governor configuration file:

```
interval 5;
setlimit cpu 5   action priority 1;
setlimit cpu 10  action priority 2;
setlimit cpu 20  action priority 3;
setlimit cpu 40  action priority 4;
setlimit cpu 80  action priority 5;
setlimit cpu 160 action priority 6;
setlimit cpu 320 action priority 8;
setlimit cpu 640 action priority 10;
setlimit cpu 1280 action priority 20;
```

As the application accumulates CPU time, the priority of its agents gets reduced.

5.14 db2batch Tool

db2batch is a tool provided with DB2 Parallel Edition V1.2. It resides in the $HOME/sqllib/misc directory and does the following:

- Reads SQL statements from a flat file
- Dynamically describes and prepares the statements
- Returns an answer set

You can specify options for db2batch to do the following:

- Control the size of the answer set
- Control the number of rows that should be sent from this answer set to an output device
- Collect the elapsed time of the SQL statement or a set of SQL statements

5.14.1 Using db2batch Tool

The syntax is as follows:

```
db2batch [-d dbname] [-f file_name]
         [-a userid/passwd]
         [-r outfile,[outfile2]]
         [-c on/off] [-i short/long]
         [-b on/off] [-o options]
         [-v on/off] [-s on/off]
         [-cli] [-h]
```

where the options are as follows:

-d Database name. If not specified, the default database name set in $DB2DBDFT will be used.

-f Input file containing SQL statements.

-a Authentication user ID/password.

-r Output file containing query results. [outfile2] will contain just the summary, but is optional. Default - stdout.

-c Automatically commit at end of each SQL statement. Default - on.

-i Elapsed time interval measurement.

- short - run time for query (default)
- long - time up to start of next query

-b Process groups of statements as blocks instead of individually. Default - off.

-o Control options:

```
r <rows out> f <rows fetch> p <perf_detail>
```

The control option can also be specified in the file where the SQL statements reside. The syntax is as follows:

```
--#SET <control option> <value>
```

-v Verbose. Sends information to stderr during query processing. Default - off.

-s Summary table. Provides summary of elapsed and CPU times with arithmetic and geometric means for those values collected. Default - on.

-cli cli mode will be used when db2batch executes. The default is embedded dynamic mode. The default mode is somewhat faster.

-h Display help text.

Note:

- All SQL statements must be terminated by a semicolon (";").
- db2batch issues its own connect and connect reset.
- PERF_DETAIL > 1 is not available in DB2 Parallel Edition V1.2.
- All statements are executed with isolation level RR.
- The maximum SQL statement size is 3999 characters.

The following are the statements you can specify in the input file for db2batch:

- All SQL statments must end with a semicolon (";"). For example:

```
SELECT * FROM ORG;
```

- Statement for comment can be specified using the syntax as follows:

```
-- comment text
```

- Statement for comment that goes to output can be specified using the syntax as follows:

```
--#COMMENT output comment text
```

- Statement for control option can be specified using the syntax as follows:

```
--#SET <control option> <value>
```

  where the possible control option and its corresponding value is:

```
   option     value       default
ROWS_FETCH   -1 to n        -1    (all rows fetched from answer set)
ROWS_OUT     -1 to n        -1    (all fetched rows sent to output)
PERF_DETAIL   0 to 1         1    (elapsed time and statement)
PAUSE                             (prompts the user to continue)
TIMESTAMP                         (generates a timestamp)
```

5.14.2 Example Using db2batch Tool

The following example will illustrate how to use db2batch to collect the elapsed time from the SQL statements and control the number of rows to be fetched from the answer set and the number of fetched rows to be sent to the output.

The table ORG has 8 rows and the table STAFF has 35 rows in the database SAMPLE. The following shows the content of the input file req.file:

```
--#SET ROWS_FETCH -1
--#SET ROWS_OUT 5
--#SET PERF_DETAIL 1
--#SET TIMESTAMP
select * from org;
--#SET ROWS_FETCH 10
--#SET ROWS_OUT 5
--#SET PERF_DETAIL 0
--#SET TIMESTAMP
select * from staff;
```

Execute db2batch using the following command:

```
db2batch -d SAMPLE -f req.file -r req.out -i long
```

The output is sent to the file req.out and the following shows its content:

```
-----------------------------------------------
--#SET ROWS_FETCH -1          1
--#SET ROWS_OUT 5
--#SET TIMESTAMP

Statement number: 1
Current Timestamp:   Sun Sept 08 17:57:24 1996

select * from org

DEPTNUMB  DEPTNAME        MANAGER  DIVISION    LOCATION
------------------------------------------------------------------
      15  New England          50  Eastern     Boston
      51  Plains              140  Midwest     Dallas
      10  Head Office         160  Corporate   New York
      38  South Atlantic       30  Eastern     Atlanta
      20  Mid Atlantic         10  Eastern     Washington


Number of rows retrieved is:        8
Number of rows sent to output is:   5

Elapsed time is:           0.469      seconds       2

-----------------------------------------------
--#SET ROWS_FETCH 10
--#SET ROWS_OUT 5
--#SET PERF_DETAIL 0           3
--#SET TIMESTAMP

Statement number: 2
Current Timestamp:   Sun Sept 08 17:57:25 1996

select * from staff

ID      NAME       DEPT    JOB    YEARS   SALARY     COMM
-----------------------------------------------------------------
    30  Marenghi       38  Mgr         5   17506.75         n/a
    20  Pernal         20  Sales       8   18171.25     612.45
    60  Quigley        38  Sales     n/a    16808.30     650.25
    10  Sanders        20  Mgr         7   18357.50         n/a
   100  Plotz          42  Mgr         7   18352.80         n/a


Number of rows retrieved is:       10
Number of rows sent to output is:  5


Summary of Results
==================

Statement #   Elapsed Time (s)  Total Application CPU Time (s)
1               0.469               Not Collected           2
2               Not Collected       Not Collected           3

Arith. mean     0.469
Geom.  mean     0.469
```

1 indicates all rows are to be fetched from answer set.

2 indicates the elapsed time for statement # 1 is collected because the default value for PERF_DETAIL is 1 which means the elapsed time for the statement is to be collected.

3 indicates that the elapsed time for statement # 2 is not collected.

5.15 DB2 Parallel Edition Database Director

DB2 Parallel Edition V1.2 provides a utility called the database director to administer your DB2 Parallel Edition database system. You can use this utility to administer the following:

- The database manager nodes
- The database manager configuration
- The resources for:
  - Database agents, such as maximum coordinating agent
  - Connection, such as the number of active local connections
  - Fast communication manager, such as free FCM request blocks
  - Sorting, such as sort heap allocated

For more information about how to install and set up the database director, refer to *DB2 Parallel Edition for AIX Administration Guide and Reference*, and the `$HOME/sqllib/Readme/$LANG/README` file.

For more information about how to use the database director, press **F1** on each window from the database director.

5.15.1 Using the Database Director

The following are examples for use with this section.

- EXAMPLE 1 illustrates an error that has occurred when starting up the Database Director on a 4-node system.

  1. Specify the name of the TCP/IP interface to be used by the Database Director on the node where you are running.

     The volume of data returned during performance monitoring can be quite large, potentially enough to flood an Ethernet connection. To avoid this situation, use the RS/6000 SP Switch, or a guaranteed bandwidth communications interface such as token ring.

     The following command displays all network interfaces available:

     ```
     netstat -i
     ```

     Specify the TCP/IP hostname to use in the DB2DD_SW_NAME environment variable:

     ```
     export DB2DD_SW_NAME=switch0
     ```

     Note: It is strongly recommended to place the setting of the DB2DD_SW_NAME environment variable in your .profile file.

2. Start up the Database Director in the background by issuing the following command:

   ```
   db2dd &
   ```

   The Database Director window appears with the icon of the database manager instances.

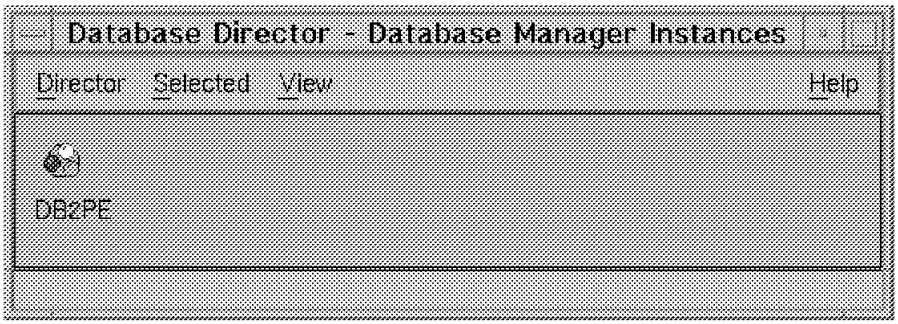


Figure 66. Database Director - Database Manager Instances

3. An error message box appears.

Figure 67. Error Message from Database Manager Instances

The possible cause of the error is either that the database manager for node 3 is not started or the Database Director daemon db2dd_D is not started on node 3.

- EXAMPLE 2 shows the steps for starting the database manager instance on a 2-node system using the database director.

  1. After database director is started, the following window appears:

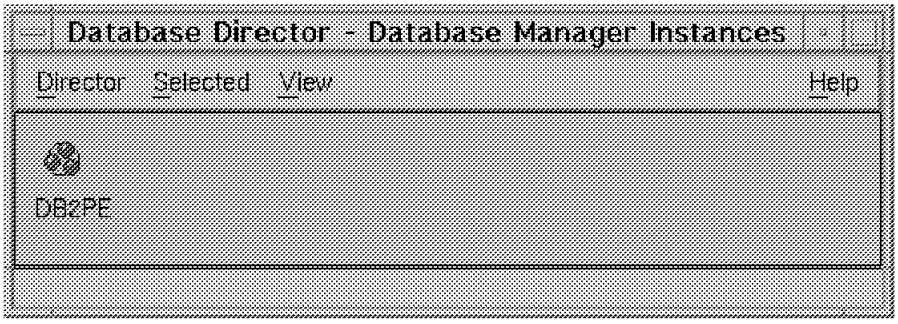


Figure 68. Database Director - Database Manager Instances

Note: The circle in the icon is dark-colored, which indicates that the database manager nodes are down at this time.

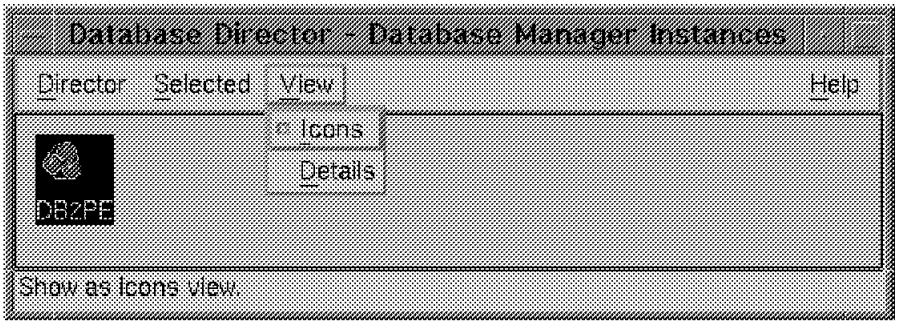


Figure 69. View Database Manager Instance

2. Click on the icon and select **Details** in the View pull-down menu and the following window will be shown:

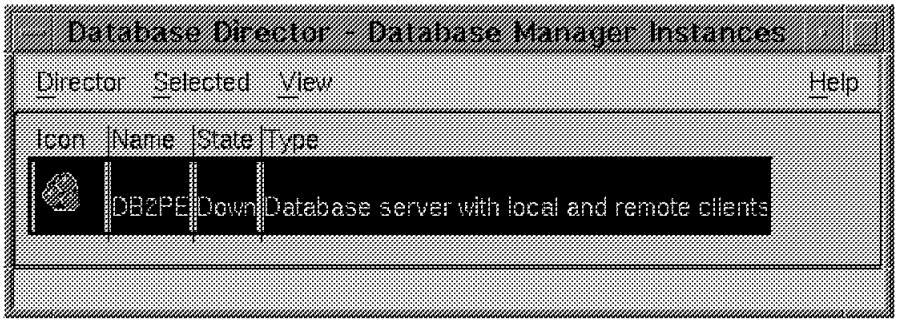


Figure 70. View Database Manager Instance in Detail

This window shows the name, the state, and the type of the instance. It indicates the instance DB2PE is down at this time.

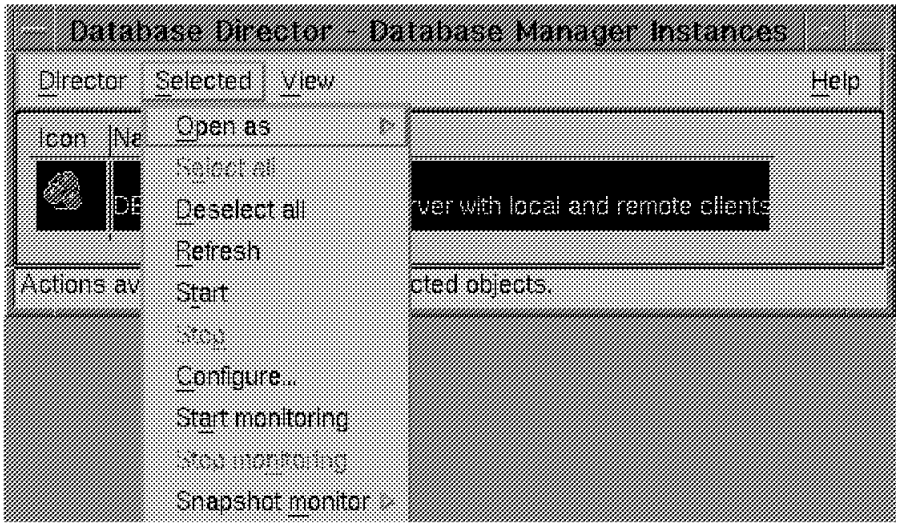


Figure 71. Start Database Manager Instance

3. Select **Start** in the Selected pull-down menu to start the database manager nodes for the instance.

   The following window shows that the state of the DB2PE instance has changed to starting:

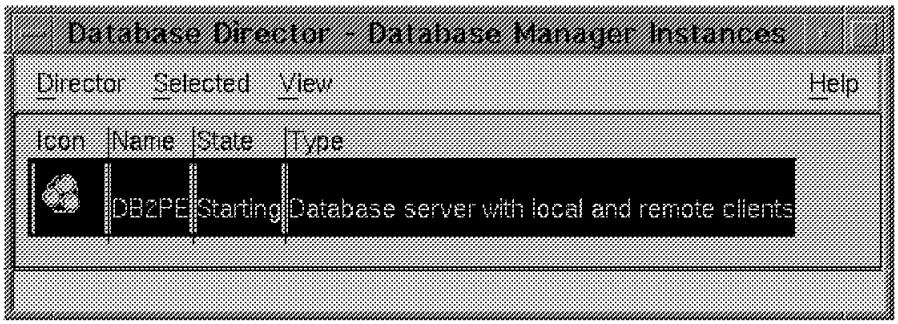


Figure 72. Starting Database Manager Instance

After all the database manager nodes are started, the state of the DB2PE instance in the following window will change to started.

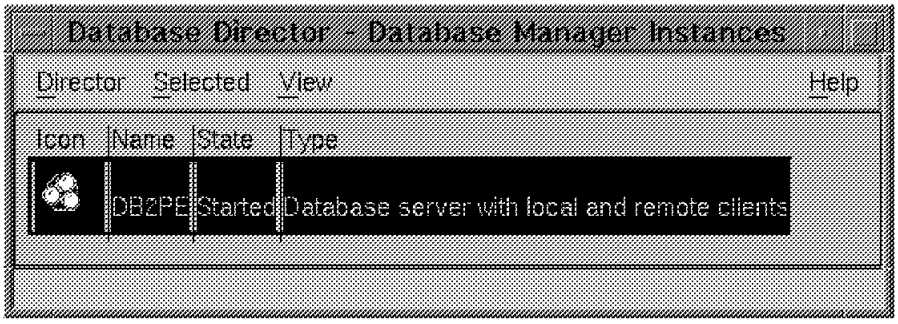


Figure 73. Database Manager Instance is Started

- EXAMPLE 3 demonstrates the steps needed to set the alert with the beep sound and message when the database manager node for the DB2PE instance is down or when the state of the database manager node is unknown.

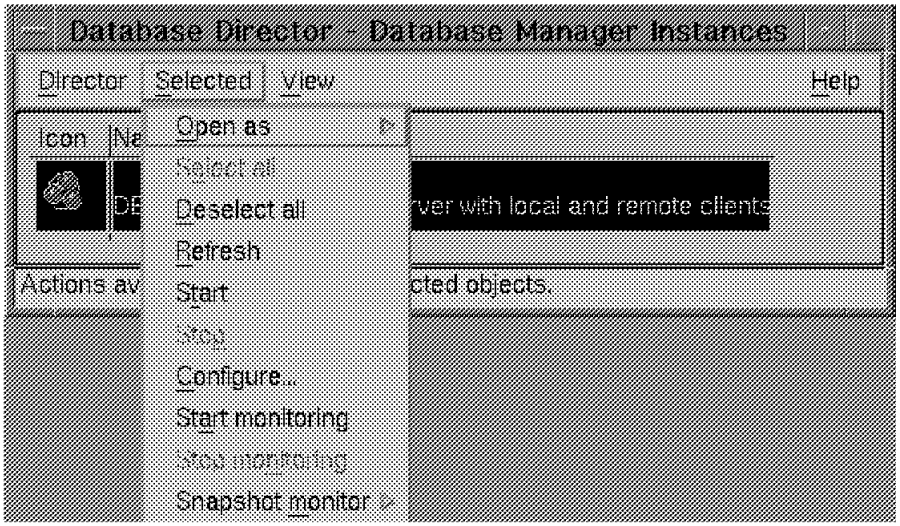


Figure 74. Open the Database Manager Instance

1. Select **Open as** in the Selected pull-down menu and the window in the following figure will be shown:

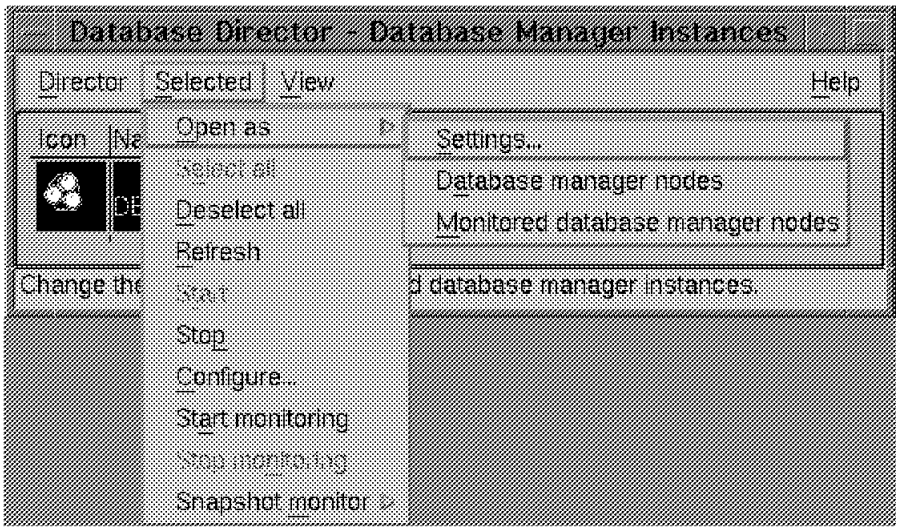


Figure 75. Open Database Manager Instance for Settings

2. Select **Settings...** in the Open as pull-down menu and the Instance settings window will be shown.

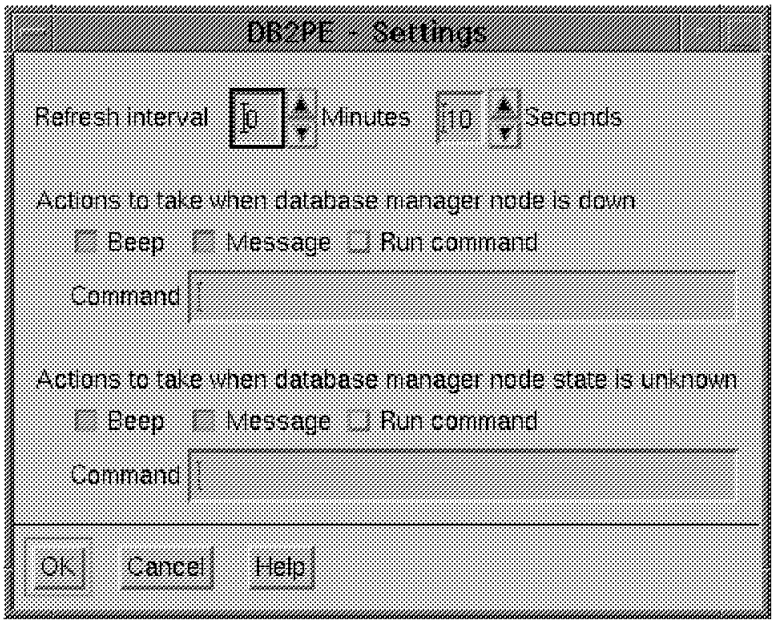


Figure 76. Database Manager Instance Settings

3. You can adjust the Refresh interval. In this example, the default of 10 seconds is chosen.
4. Under Actions to take when database manager node is down, select **Beep** and **Message**.
5. Under Actions to take when database manager node state is unknown, select **Beep** and **Message**.
6. Click on **OK**.

The alert is set. When either the database manager node is down or the state of the database manager node is unknown, the system will generate a beep sound and the following message window will be displayed:

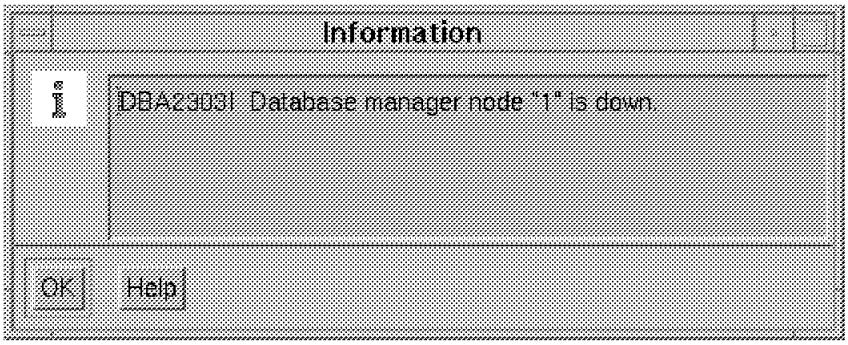


Figure 77. Database Director Alert Message

- EXAMPLE 4 shows the steps to start the database monitor and set the alert for the beep sound and message when the number of coordinating agents is greater than 3 on database manager node 0 for database instance DB2PE.

  1. From the window shown in Figure 74 on page 220, select **Open as** in the Selected pull-down menu and the window in Figure 75 on page 221 will be shown.
  2. From the window shown in Figure 75 on page 221, select **Database manager nodes** from the Open as pull-down menu. The database manager nodes window is displayed.

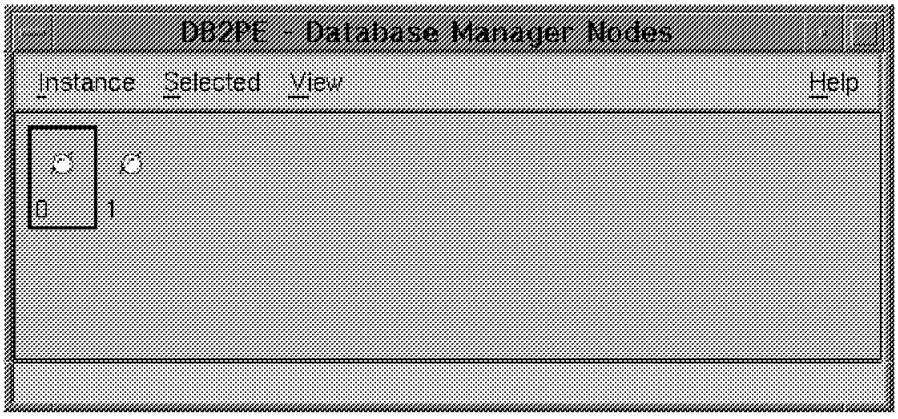


Figure 78. Database Manager Nodes for Database Manager Instance

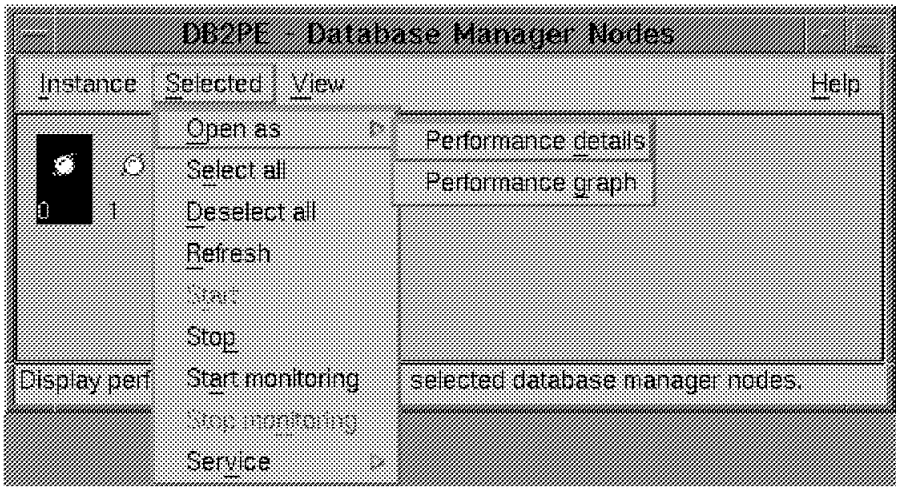


Figure 79. Open Database Manager Nodes for Instance

3. Select **Open as** from the Selected pull-down menu. Then select **Performance details** from the Open as pull-down menu. The performance details window is displayed.

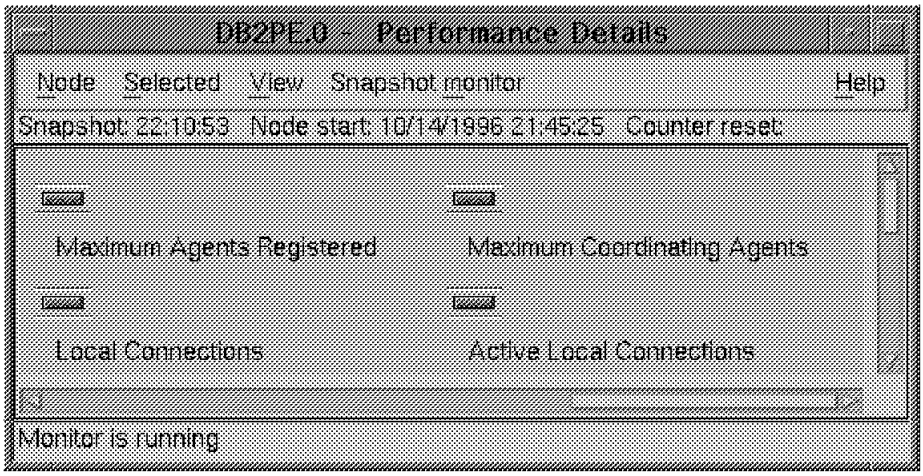


Figure 80. Performance Details for Instance on Database Manager Node

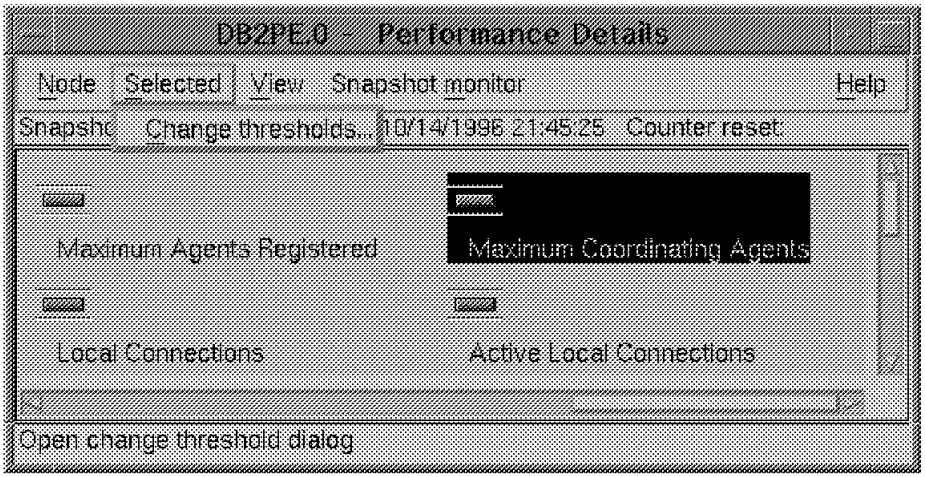


Figure 81. Change Thresholds for Instance on Database Manager Node

4. Scroll to the right to search for Maximum Coordinating Agents. Once found, click on **Maximum Coordinating Agents**, then select **Change thresholds** from the Selected pull-down menu. The Maximum Coordinating Agents - Change Thresholds window is displayed.

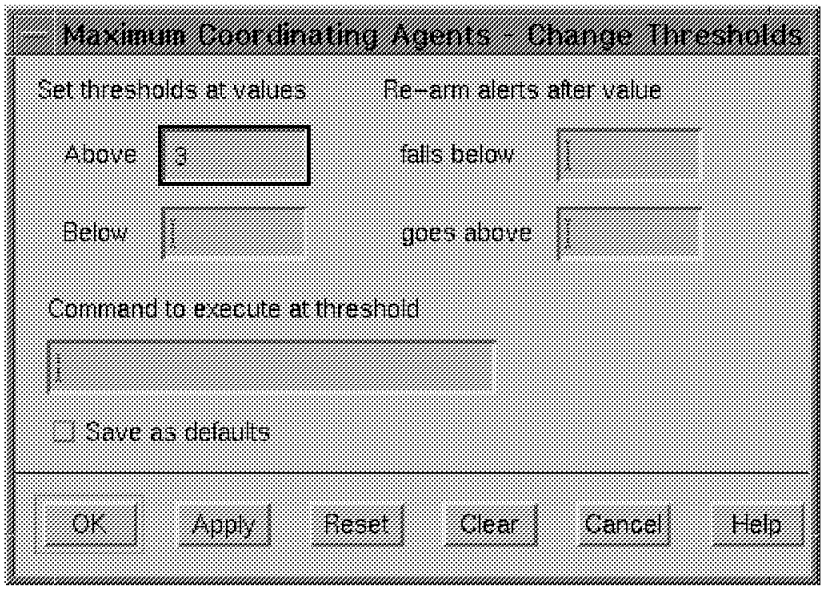


Figure 82. Maximum Coordinating Agents - Change Thresholds

5. Enter **3** in the Above field for the value of the threshold.
6. Click on **OK**.

Go back to the window in Figure 80 on page 223.

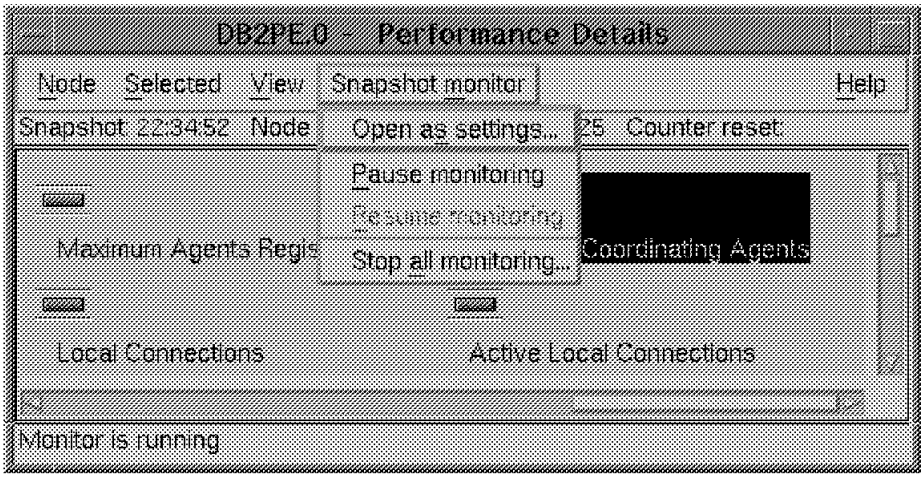


Figure 83. Set Actions for Snapshot Monitor

7. Click on **Maximum Coordinating Agents**, then select **Open as settings** from the Snapshot monitor pull-down menu. The Snapshot monitor settings window is displayed.

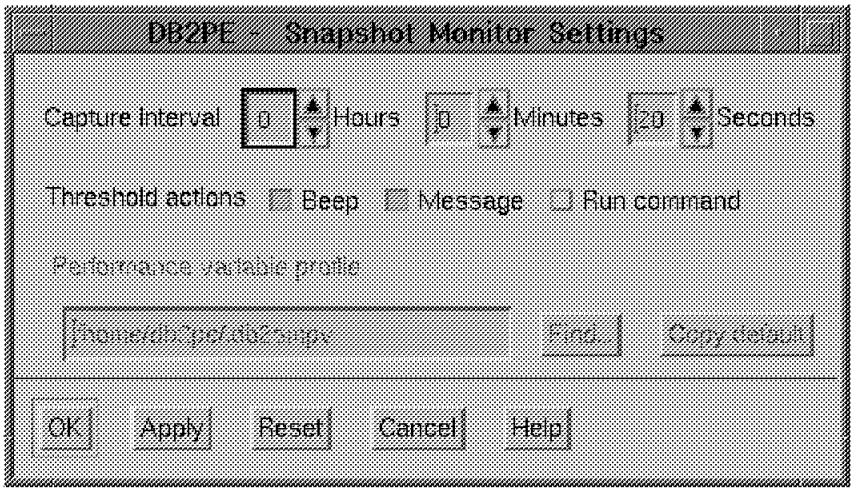


Figure 84. Snapshot Monitor Settings

8. You can adjust the `Capture Interval`. In this example, the default is 20 seconds.
9. Select **Beep** and **Message** for the threshold actions.
10. Click on **OK**.

    When the number of coordinating agents is equal to the value of the threshold which was set for the maximum coordinating agents, the alert will trigger and the system will generate a beep sound and an alert message box.

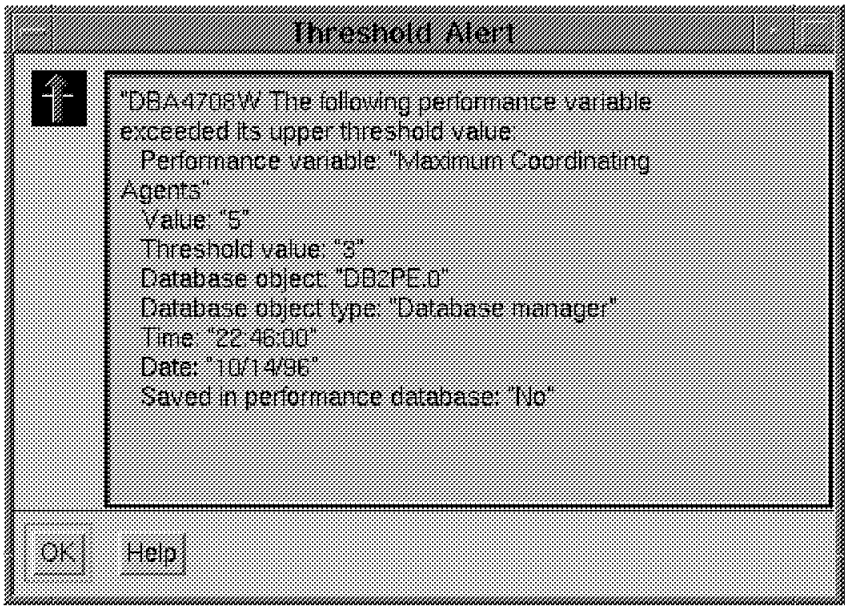


Figure 85. Alert Message from Snapshot Monitor

Chapter 6. Installation and Configuration

This chapter describes the steps required to install and configure a parallel database environment using DB2 PE. There is also a section on HACMP configuration and a section on the DRDA application server feature. The chapter is divided as follows:

- Installation procedure
- Configuration procedure
- HACMP configuration
- DRDA application server feature

The first section looks at the basic system setup and installation tasks necessary for DB2 PE. The second section explains how to configure the system, using the minimum steps required to check that everything is working correctly. There are also hints on an RS/6000 SP environment. The third section describes two configurations for the HACMP software: idle standby and rotating standby. Finally, the section describes the DRDA application server feature in DB2 Parallel Edition V1.2

6.1 Installation Procedure

In this section, we describe the steps needed to install DB2 PE. The section is divided in two subsections:

- Pre-installation tasks
- Installation tasks

For each of these subsections, we first give an overview of the steps required, followed by a detailed description of a method for doing each of these steps.

6.1.1 Hardware Environment

An RS/6000 SP containing 8 wide nodes was configured as the DB2 PE environment. The machines were connected via Ethernet, token-ring and HPS. See Figure 86 on page 228 for a diagram showing this setup.

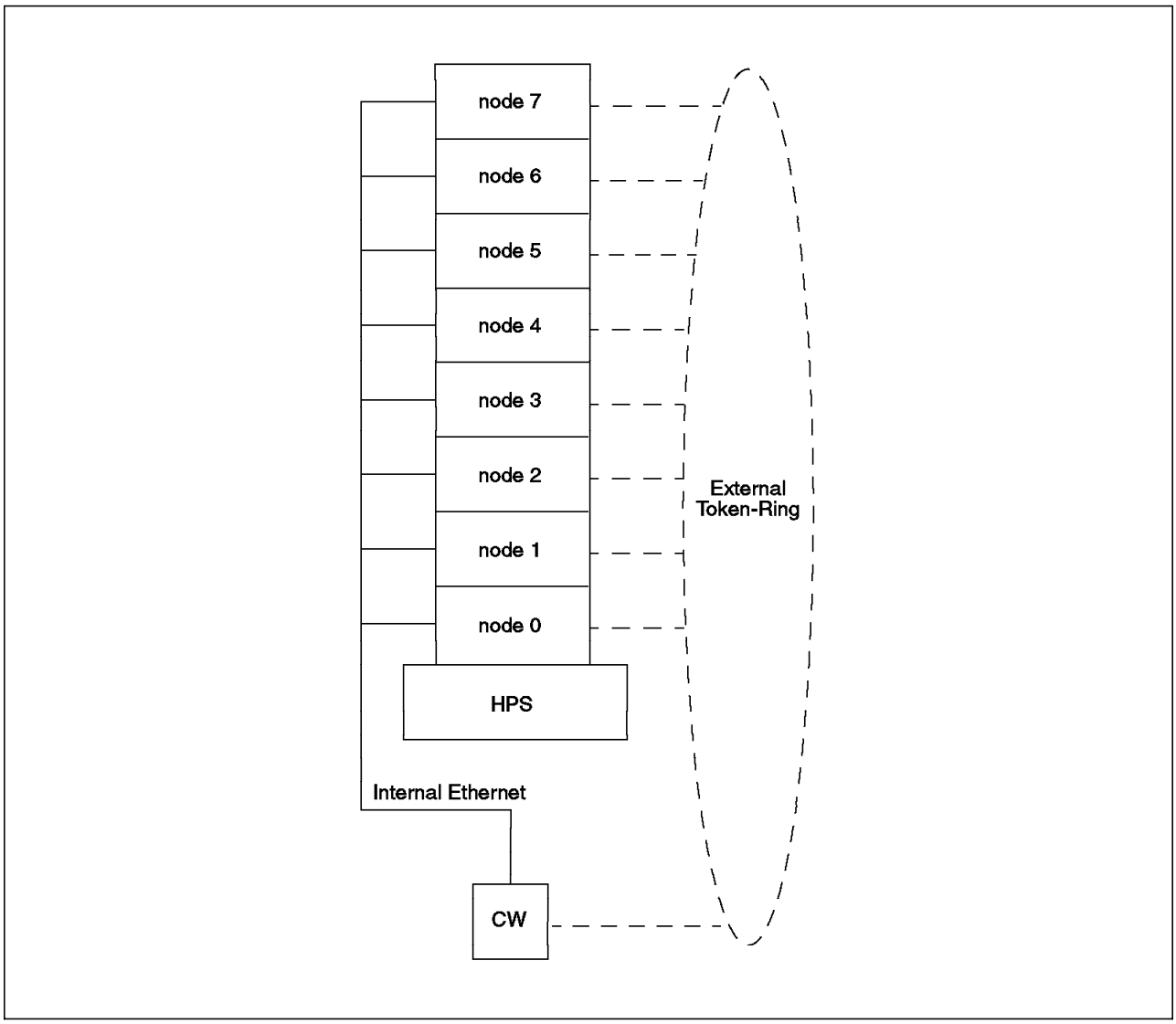


Figure 86. Hardware Environment

6.1.2 Pre-Installation Tasks

The system administrator, as root user, should complete the following tasks on every node:

1. Create a group for DB2 PE.
2. Create a user for DB2 PE.
3. Set up the home directory for DB2 PE.
4. Decide on how to distribute the DB2 PE software.
5. Increase the maximum number of processes per user.
6. Provide sufficient paging space.
7. Configure syslog.
8. Tune TCP/IP network parameters.
9. Create a local file system to contain a DB2 PE database

10. Give ownership of the database directory of that local file system to the instance owner of DB2 PE.

In the following section, group names, user names, ID numbers, and so on are given as examples. These will be written in italics, for example *db2pe*.

Two different system environments will be considered throughout this section. Many other configurations are also possible. The following are the two scenarios described:

- A setup which uses no network management software; so each operation must be replicated on every node
- An NIS and NFS environment

The first scenario describes a minimal environment which most administrators should be able to easily recreate. The number of physical machines in your parallel environment will help to guide you as to which scenario is more appropriate.

Note: On an RS/6000 SP machine, you could use file collections instead of NIS. The description and management of file collections are not covered in this document. Also, in this section, the text refers to running a command on all nodes. The RS/6000 SP system provides the distributed shell command, dsh, which will assist in running these commands. Examples of dsh are:

```
dsh -a command
dsh -w host1, host2, ..., hostn command
```

The first command is sent to all the nodes of an RS/6000 SP. The list of all nodes of the RS/6000 SP is stored in the System Database Repository. This command must be issued from an RS/6000 SP node or from the control workstation. The second example sends the command to the specified nodes. For more information on dsh, refer to the *Scalable POWERparallel 2 Administration Guide*.

6.1.3 Create Group for DB2 PE Instance Owner

On every node, the system administrator, as the root user, has to create a group for the instance owner, *db2pe*. This group must have the same group ID on all machines. Any member of this group will have SYSADM privileges for the DB2 PE instance. The following steps are involved:

1. Execute the command:

   ```
   mkgroup db2grp
   ```

2. Check the group ID number on all machines by running the command:

   ```
   lsgroup db2grp
   ```

3. The group ID must be consistent across all nodes. The number must be unused on all machines. Execute the following command to update the group ID:

   ```
   chgroup id=702 db2grp
   ```

   Note: The command will give a warning message which can be ignored.

If you are using NIS, the following steps should be followed on the NIS master server:

1. Insert a new entry into the /etc/group file. The entry should be similar to the following:

   ```
   db2grp::702:db2pe
   ```

2. Refresh the NIS tables by running:

   ```
   /etc/yp/make
   ```

   Note: You may wish to wait until after the user has been created before running this command.

6.1.4 Create Instance Owner for DB2 PE

The system administrator, as root user, must now create a user who will be the instance owner. We will use the user *db2pe* with primary group *db2grp* on every node. The user ID number must be consistent across all nodes, and unique on each node.

1. Execute the command:

   ```
   mkuser id=802 pgrp=db2grp db2pe
   ```
   or
   ```
   mkuser id=802 pgrp=db2grp home=home-dir-path db2pe
   ```

   Specifying:

   - User name = *db2pe*
   - User ID = 'unique number' (*802* in our example)
   - Primary group = *db2grp*
   - Home directory if the default, /u/db2pe, is not to be used

2. To give the instance owner, *db2pe*, a password, execute the command:

   ```
   passwd db2pe
   ```

Users on an RS/6000 SP can use the `spmkuser` command to create the user, and to automatically export the home directory using amd:

```
spmkuser id=802 pgrp=db2grp db2pe
```

If you are using NIS, you must do the following on the NIS master server:

1. Add an entry to file /etc/passwd, for example:

   ```
   db2pe::802:702:Comment:/u/db2pe:/bin/ksh
   ```

   Here, `Comment` is an optional field where you may supply user comments or the name of the person responsible for this user ID. This field may be left blank.

2. Refresh the NIS tables by running the command:

   ```
   /etc/yp/make
   ```

3. Give the user ID a password in NIS by issuing the command: can be set by issuing the command:

   ```
   yppasswd db2pe
   ```

6.1.5 Home Directory for the Instance Owner

There must be only one home directory for the DB2 PE instance user. This home directory must be shared by all hosts. This may be done via NFS, AFS or DFS. We will only describe the use of NFS.

1. Start NFS by executing the following command on all nodes:

   ```
   /usr/etc/mknfs -B
   ```

2. The directory is exported by executing the following command on the node which will own the directory:

   ```
   mknfsexp -d/u/db2pe -t rw -c 'host1,host2,...,hostn' -B
   ```

3. This directory must then be mounted on all other nodes. This is done by entering the following command on all other nodes:

   ```
   /usr/etc/mknfsmnt -f /u/db2pe -d /u/db2pe -h host0
            -n -B -A -t rw -w bg -Y -Z -X
   ```

Note: Products exist which perform automatic NFS mounts and unmounts. These can be used to improve NFS performance. Two products which provide this service are amd and automount. If your hardware type is an RS/6000 SP, both of these tools should be available for use. To do this, the following steps are required:

1. Update the following file:

   - For amd, update /etc/amd/amd-maps/amd.u by adding the following line:

     ```
     db2pe  host==sp2n1;type:=link;fs=/home/sp2fs/db2pe \
              host!=sp2n1;type:=nfs;rhost:=sp2n1;rfs:=/home/sp2fs/db2pe
     ```

   - For automount, update /etc/auto/auto.u by adding the following line:

     ```
     db2pe  -rw,hard,bg,intr    sp2n1=/home/sp2fs/db2pe
     ```

2. Propagate the maps across the nodes either manually on all nodes or by running the following command on the NIS master server if NIS is set up to spread the maps:

   ```
   /etc/yp/make
   ```

3. Refresh the maps:

   - For amd, issue on every node the following command:

     ```
     /etc/amd/refresh_amd
     ```

   - For automount, kill and restart the automount daemon using the following method:

     ```
     ps -ef | grep automount
     kill PID Number
     /usr/sbin/automount
     ```

     where PID Number is the AIX PID returned by the ps command.

6.1.6 Decide on the Distribution of the DB2 PE Software

You may decide to install DB2 PE either locally on every node, or just once on a master node, and mount the software directories remotely. Release 1.2 of DB2 PE installs into the directory /usr/lpp/db2pe_01_02. This directory must either be installed onto every node or network mounted across the nodes. NFS mounting the software may be an easier option than installing on every node, but performance may suffer.

In an RS/6000 SP environment, you can create an image of a node after the installation of DB2 PE, and replicate this on other nodes. This procedure, standard on the RS/6000 SP machines, makes the installation process easier. For details on this procedure, please refer to *SP2 Administration Guide*.

6.1.7 Increase the Number of Processes Per User

It is necessary to change the maximum number of processes allowed per user. This is found in the MAXUPROC parameter. The recommendation is to set MAXUPROC to 500. The following command must be executed on all nodes:

```
/etc/chdev -l sys0 -a maxuproc=500
```

6.1.8 Provide Sufficient Paging Space

A paging space is fixed disk storage for information that is resident in virtual memory, but is not currently being accessed. A paging space, also called swap space, is a logical volume with the attribute type equal to paging. This type of logical volume is referred to as a paging-space logical volume, or simply paging space. When the amount of real memory in the system is low, programs or data that have not been used recently are moved from real memory to paging space to release real memory for other activities.

DB2 PE typically requires paging space that is at least twice the size of real memory. The `lsps -a` command can be used to display all paging spaces on a system:

```
Page Space  Physical Vol  Vol Group  Size   %Used  Active  Auto  Type
paging01    hdisk2         rootvg    40MB    90     yes    yes    lv
paging00    hdisk1         rootvg    40MB    90     yes    yes    lv
hd6         hdisk0         rootvg    48MB      90   yes    yes    lv
```

And the `lscfg -l mem*` command can be used to display the amount of real memory installed in a system:

```
  DEVICE           LOCATION          DESCRIPTION
  mem0             00-0A             8 MB Memory SIMM
  mem1             00-0B             8 MB Memory SIMM
  mem2             00-0C             8 MB Memory SIMM
  mem3             00-0D             8 MB Memory SIMM
  mem4             00-0E             8 MB Memory SIMM
  mem5             00-0F             8 MB Memory SIMM
  mem6             00-0G             8 MB Memory SIMM
  mem7             00-0H             8 MB Memory SIMM
```

The chps, mkps, rmps, and swapon commands are used to manage paging space, but their use is beyond the scope of this book. Please refer to the appropriate AIX system documentation.

6.1.9 Configure syslog

DB2 PE uses the system logger (SYSLOG) to log error and warning conditions. Entries are added to the SYSLOG based on priority and what facility caused the error or warning condition. DB2 PE is represented by the facility called 'user'. Priority refers to the urgency of the message, and they are as follows in highest to lowest priority:

- Emergency
- Alert
- Critical
- Error
- Warning
- Notice
- Information
- Debug

To begin logging DB2 PE error and warning conditions in the SYSLOG, you must start logging for the facility of type 'user'. This is accomplished by editing the /etc/syslog.conf file. For example, add the following line to this file:

```
user.warn   /var/tmp/syslog.out
```

Note: You must have root authority to do this. Also, if this file grows quickly, you may have to reduce its size periodically.

Next the file must be created:

```
touch /var/tmp/syslog.out
```

Finally, the syslog daemon must be sent a signal of '1' to extract the latest information from the /etc/syslog.conf file. This can be accomplished by entering the following command:

```
refresh -s syslogd
```

6.1.10 Tuning TCP/IP Network Parameters

The settings for the following parameters should be reviewed:

1. Maximum Transmission Unit (MTU) size
2. TCP/IP network options
3. SP Switch device driver

6.1.10.1 Maximum Transmission Unit (MTU) Size

The maximum MTU size for the switch network is 65520, but setting it this high may reduce Ethernet performance. The recommended MTU size is 16394. This will give a middle ground for performance on the switch and the Ethernet in terms of retransmissions.

6.1.10.2 TCP/IP Network Option

Setting the TCP/IP network option parameters (sometimes referred to as the **no** parameters) may affect the names adapter's performance. In other words, setting the values for the switch will focus on the switch's performance at the possible expense of the Ethernet performance.

	Thin Node Ethernet	Wide Node Ethernet	Switch
lowclust	100	100	100
thewall	6144	6144	6144
mb_cl_hiwat	1220	1220	1220
rfc1323	1	1	1
tcp_sendspace	118784	221184	655360
tcp_recvspace	118784	221184	655360
udp_recvspace	118784	221184	655360
udp_sendspace	59392	110592	327680
sb_max	237568	442368	1310720

The current network options can be displayed with the `no -a` command. Network options are usually set in the `/etc/rc.net` file on RS/6000 systems, and in the `/tftpboot/tuning.cust` file on RS/6000 SP systems.

6.1.10.3 SP Switch Device Driver

The device driver for the SP Switch (not the previous generation High Performance Switch) has two parameters, send and receive pool sizes, that should be increased from their default of 512 KB for use with DB2 PE. Setting the pool sizes between 1 MB to 2 MB would be a good starting point:

```
/usr/lpp/ssp/css/chgcss -l css0 -a spoolsize=2097152
/usr/lpp/ssp/css/chgcss -l css0 -a rpoolsize=2097152
```

Note: The system must be rebooted to activate the change.

The `lsattr -E -l css0` command can be used to display the current settings, and the `vdidl3 -i` command can be used to display the buffer statistics for the device driver.

6.1.11 Create File System for Database

The database data should reside locally on each node. To assist in the management of your system, we suggest that you create a separate file system for the database. To help to calculate the size of this filesystem, The following equation may be helpful:

```
T = total size (in MB) of an equivalent serial DB2 database                    )
(including indexes)
N = number of nodes

Size = ( T / N + 3 MB ) * 1.2 * 2048
```

To create the filesystem, run the following command on every node:

```
/etc/crfs -v jfs -g datavg -a size=Size -m/database -A yes -p rw -t no
```

where *Size* is the size in 512 blocks as calculated above. To assist in determining the size of a serial database, refer to *DB2 Administration Guide*.

6.1.12 Change Ownership of Database File System

The last step of the pre-installation tasks is to give ownership of the database directory in the file system you created to the instance owner, *db2pe*. We recommend that a subdirectory is first created within the filesystem, for example */database/db2pe*. This subdirectory will contain the database if created by user *db2pe*. Setting permissions on this directory only will allow other users to also use the same file system, without causing ownership conflicts. To do this, first create the subdirectory with the following command on all nodes:

```
mkdir /database/db2pe
```

Now, update the ownership of the directory by executing the following command on all nodes:

```
chown db2pe.db2grp /database/db2pe
```

6.1.13 Installation Tasks

If you decided to install on all nodes, the following section must be replicated over all nodes. If you decide to remotely mount the software directory, it should be run only on the master node, and this directory should be remotely mounted on all nodes. All commands should be run as root user.

6.1.14 Software Installation

The software is installed using the standard installp process. The easiest method of installation is via the SMIT install_latest fastpath. The following software features will be available on the media:

- db2pe_01_02.db2
- db2pe_01_02.sna
- db2pe_01_02.dd
- db2pe_01_02.drda
- db2pe_01_02.msg.De_DE
- db2pe_01_02.msg.de_DE
- db2pe_01_02.msg.Es_ES
- db2pe_01_02.msg.es_ES
- db2pe_01_02.msg.Ja_JP
- ipfx.Runtime
- ipfx.nls.%L
- db2pe_01_02.doc.%L.ipfx
- db2pe_01_02.doc.%L.pscript

The minimum software installation required:

- db2pe_01_02.db2

Note: If you are installing db2pe_01_02.db2 on AIX version 4.1.3, you must apply PTF U440406 to AIX. This PTF ensures that DB2 Parallel Edition can function correctly on AIX version 4.1.3.

6.1.15 Software Distribution

If you decide to install the software on only one node, the directory /usr/lpp/db2pe_01_02 (on release 1.2) must now be made available on all other nodes. If you have multiple networks connecting your nodes, it may be better to reserve the fastest network exclusively for the use of the DB2 PE product, and use a slower network for NFS. The distribution of the software over NFS is done with the following commands:

1. The directory is exported by issuing the following command on the node on which the software has been installed:

   ```
   mknfsexp -d/usr/lpp/db2pe_01_02 -t ro -B
   ```

2. This directory must then be mounted on all other nodes. This is done by issuing the following command on all other nodes:

   ```
   /usr/etc/mknfsmnt -f /usr/lpp/db2pe_01_02 -d /usr/lpp/db2pe_01_02
            -host host0 -n -B -A -t ro -w bg -Y -Z -X
   ```

6.2 Configuration

The steps involved in configuration are as follows:

1. Create an instance.
2. Create the db2nodes.cfg file.
3. Reserve the service ports for DB2 PE, and the Database Director.
4. Modify the .profile file for the instance owner.
5. Allow Remote Commands.
6. Start the Database Director daemon.

6.2.1 Create an Instance

As root, create an instance by running the command:

```
/usr/lpp/db2pe_01_02/instance/db2instance db2pe
```

This is very similar to the serial DB2/6000 and creates the system database directory for the *db2pe* user, *$HOME/sqllib*. This directory is very similar to the serial DB2/6000, and it contains configuration files, links to executables, logs, and so on. It also contains the file *$HOME/sqllib/sqldbdir/sqldbins* which is used to synchronize and recover database creation across all nodes. This file is called the System Intention File.

6.2.2 Create the db2nodes.cfg File

This file must be created to tell DB2 PE which hosts will be part of the instance. This operation should be executed as the instance owner, db2pe. The *$HOME/sqllib/db2nodes.cfg* file contains one line per node. Each line has the following fields:

- Node Number

- Node Hostname
- Port Number (Optional)
- Net Name (Optional)

For more details, see 3.3, “Parallel Database Nodes” on page 45. An example of this file is:

```
0   host0   0   switch0
1   host1   0   switch1
2   host2   0   switch2
3   host3   0   switch3
4   host4   0   switch4
5   host5   0   switch5
6   host6   0   switch6
7   host7   0   switch7
```

6.2.3 Reserve the Service Ports

The root user should now update the /etc/services file. This is used to reserve port numbers on machines and it could be shared via NIS. All nodes which were included in the db2nodes.cfg file should reserve ports. In addition, a port should be reserved for the Database Director. The additional lines added to the /etc/services file will be as follows:

```
DB2DD_db2pe     5499/tcp
DB2_db2pe       5500/tcp
DB2_db2pe_END   5507/tcp
```

6.2.4 Modify Login Environment for the Instance Owner

As the instance owner, the following entry should be added to the .profile file so that every login will include the DB2 PE environment:

```
. $HOME/sqllib/db2profile
```

This sets the following environment variables:

- DB2INSTANCE to $USER (for example, db2pe)
- PATH to $PATH:$HOME/sqllib/adm:$HOME/sqllib/bin:$HOME/sqllib/misc

The new profile should now be executed by running the command:

```
. .profile
```

6.2.5 Allow Remote Commands

As the instance owner, create a $HOME/.rhosts file. This file should have one entry per node, and its ownership bits must be set to 600. Every record has the following two entries:

- Hostname
- User ID

Example of the .rhosts file:

```
host0   db2pe
host1   db2pe
host2   db2pe
host3   db2pe
host4   db2pe
host5   db2pe
host6   db2pe
host7   db2pe
```

The file permissions are updated by using the command:

```
chmod 600 .rhosts
```

The /etc/hosts.equiv file can be used to authorize remote commands on a system-wide basis: Example of the /etc/hosts.equiv file:

```
host0   +
host1   +
host2   +
host3   +
host4   +
host5   +
host6   +
host7   +
```

Note that DB2 PE requires either a .rhosts, or /etc/hosts.equiv file. The Kerberos equivalents that are shipped with the AIX Parallel System Support Programs will not work with DB2 PE.

6.2.6 Start the Database Director Daemon

To gather the data from the database manager nodes that make up a DB2 Parallel Edition Database, a daemon must be run for each database manager instance (DB2INSTANCE) on each database manager node that will be monitored. A file called ″rc.db2dd″ is created in the ″usr/lpp/db2pe_01_02/cfg″ directory at installation time. In this directory, this file is read only.

To enable the Database Director autorestart, do the following on each node as root:

1. Copy the ″/usr/lpp/db2pe_01_02/cfg/rc.db2dd″ file to ″/etc/rc.db2dd_*instance*″.
2. Change to the ″/etc″ directory.
3. Change the file permissions to make the file write enabled and executable.
4. Edit the file to specify the instance that you want to monitor.
5. Add the file to the ″/etc/inittab″ file so that the daemon will be started after a system restart.

```
cp /usr/lpp/db2pe_01_02/cfg/rc.db2dd /etc/rc.db2dd_db2pe
cd /etc
chmod 744 db2dd_db2pe
vi db2dd_db2pe
mkitab ″db2dd_db2pe:2:respawn:/etc/rc.db2dd_db2pe >/dev/console 2>&1″
```

The ″/etc/rc.db2dd_db2pe″ should be as follows:

```
#!/bin/sh
# Initialize the Database Director Daemon (db2dd_D) for the instance
#
echo 'Initializing DB2 Database Director Daemon'
# Change db2instancename to the name of the database manager instance
# to be monitored.

while [ 1 ]
do
   if [ -f /u/db2pe/sqllib/bin/db2dd_D ]
   then
      su - db2pe ″-c db2dd_D > /dev/console 2>&″
      exit
   else
      sleep 10
   fi
done
```

6.3 Database Management

All operations should be run as the instance owner, db2pe.

6.3.1 Starting a DB2 PE Instance

As db2pe, run the following command to start DB2 PE:

```
db2start
```

6.3.2 Creating a Database

The command to create a database called *dss* in the directory /database is:

```
db2 -v ″create database dss on /database″
```

For information on the files created, refer to Chapter 3, “Concepts and Data Placement” on page 35.

6.3.3 Creating a Nodegroup

To create a nodegroup named ng4 over four nodes (0, 1, 2, 3), execute the following commands:

```
db2 connect to dss
db2 create nodegroup ng4 on node(0,1,2,3)
db2 connect reset
```

For more details on nodegroups, see 3.7.1, “Nodegroups” on page 60.

6.3.4 Creating a Table

The command to create a table is identical to that used in serial DB2/6000, with the optional additions of defining a partitioning key and a nodegroup. Examples of the commands used to do this are:

```
db2 connect to dss
db2 create table lineitem (l_orderkey integer    not null, \
                           ...        ...        ...
                           l_comment  char(59)   not null) in ng4
db2 connect reset
```

This creates a table called lineitem in nodegroup ng4. For more information on this process, please see 3.7.5, “Partitioning Key” on page 65.

6.3.5 Stopping a DB2 PE Instance

As the instance owner, issue the following command to stop DB2 PE:

```
db2stop
```

6.4 HACMP Configurations

When using HACMP/6000 with DB2 PE and your system configuration experiences a failure, the HACMP software will execute. By using HACMP you can take over resources such as disk. A machine could serve as a standby processor in the event of a processor failure. The HACMP software will cause the following to occur:

- Takeover of the network address
- Takeover of some filesystems (user-defined)
- Execute a user-defined script

For more information regarding of HACMP implementation for DB2 Parallel Edition, refer to the ITSO Redbook *Backup, Recovery and Availability with DB2 Parallel Edition on RS/6000 SP*, SG24-4695-00.

The following configurations were tested with HACMP/6000:

- Idle Standby
- Rotating Standby

All configurations were done on a set of three machines. The hardware setup was identical for all of them, and is shown in Figure 87 on page 241. Two machines, Host 0 and Host 2, form an HACMP cluster. They both have access to a shared disk containing the following filesystems:

- /u/db2pe
- /usr/lpp/db2pe_01_02
- /database

The first two filesystems are NFS exported, to be mounted on the third machine, Host 1. Host 0 and Host 2 both have two token-ring adapters to attach to the network. This is to allow, for example, Host 0 to own the IP addresses for Host 0 and Host 2 at the same time. Host 1 does not require multiple adapters since it is not running HACMP. Hosts 0 and 2 have an RS232 serial connection to prevent a false takeover in the event of a token-ring network failure. This configuration is obviously flawed from a high availability point of view because Host 1 does not have any backup; however, it allows the

possibilities of HACMP with DB2 PE to be explored with a minimal configuration.

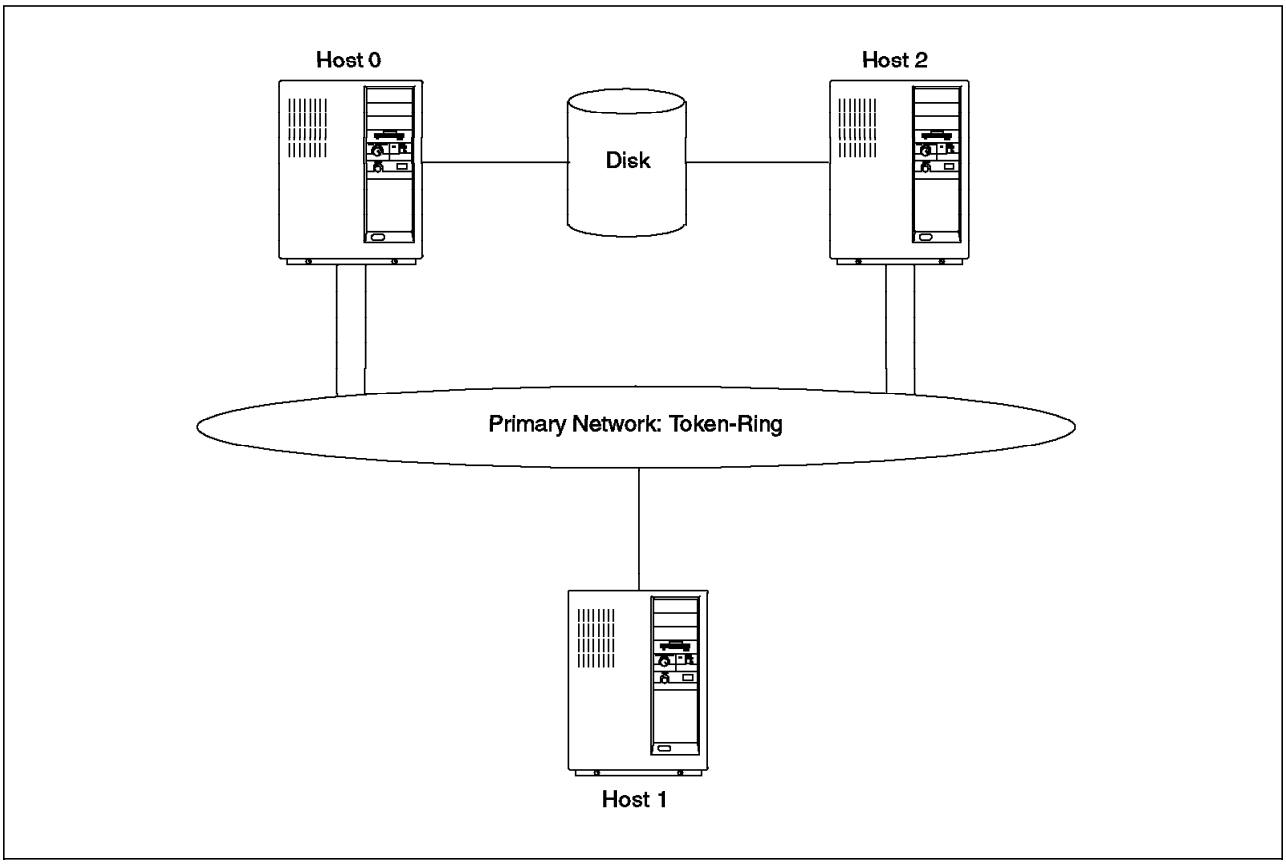


Figure 87. HACMP Test Setup

The following sections describe each of the configurations in detail.

6.4.1 HACMP Idle Standby

In the Idle Standby configuration, Host 0 is set up to hold Node 0 of a DB2 PE instance, and Host 1 contains Node 1. Host 2 is a standby machine, and if Host 0 fails, it will take over the resources for Node 0 and restart DB2 PE. The HACMP cluster configuration is shown below:

```
Cluster Description of Cluster db2pe
Cluster ID: 1
There were 2 networks defined : net1, net2
There are 2 nodes in this cluster.

NODE host0:
        This node has 2 service interface(s):

        Service Interface host0:
                IP address:     9.3.1.88
                Hardware Address:       0x42005a4f4165
                Network:        net1
                Attribute:      public

        Service Interface host0 has a possible boot configuration:
                Boot (Alternate Service) Interface: host0_boot
```

```
                IP address:     9.3.1.188
                Network:        net1
                Attribute:      public

        Service Interface host0 has 1 standby interfaces.
                Standby Interface 1: host0_standby
                IP address:     9.3.4.88
                Network:        net1
                Attribute:      public


        Service Interface host0_tty1:
                IP address:     /dev/tty1
                Hardware Address:
                Network:        net2
                Attribute:      serial

        Service Interface host0_tty1 has no standby interfaces


NODE host2:
        This node has 2 service interface(s):

        Service Interface host2:
                IP address:     9.3.1.78
                Hardware Address:
                Network:        net1
                Attribute:      public

        Service Interface host2 has 1 standby interfaces.
                Standby Interface 1: host2_standby
                IP address:     9.3.4.78
                Network:        net1
                Attribute:      public


        Service Interface host2_tty0:
                IP address:     /dev/tty0
                Hardware Address:
                Network:        net2
                Attribute:      serial

        Service Interface host2_tty0 has no standby interfaces



Breakdown of network connections:

Connections to network net1
        Node host0 is connected to network net1 by these interfaces:
                host0_boot
                host0
                host0_standby

        Node host2 is connected to network net1 by these interfaces:
                host2
                host2_standby
```

```
Connections to network net2
        Node host0 is connected to network net1 by these interfaces:
                host0_tty1

        Node host2 is connected to network net1 by these interfaces:
                host2_tty0
```

The resource configuration for this setup is shown below:

```
Resource Group Name               db2r
Node Relationship                 cascading
Participating Node Name(s)        host0 host2
Service IP Label                  host0
Filesystems                       /database /u/db2pe
                                  /usr/lpp/db2pe_01_02
Filesystems to be exported        /u/db2pe /usr/lpp/db2pe_01_02
Filesystems to be NFS mounted
Volume Groups
Concurrent Volume Groups
Disks
Application Servers               db2pe
Miscellaneous Data
Inactive Takeover                 false
9333 Disk Fencing                 false


Run Time Parameters:

Node Name                         host0
Debug Level                       high
Host uses NIS or Name Server      false

Node Name                         host2
Debug Level                       high
Host uses NIS or Name Server      false
```

An application server, db2pe, is defined to the cluster. This uses two scripts, startpe and stoppe to control the DB2 PE environment. The startpe script is shown below:

```
#!/bin/ksh

su - db2pe -c db2start
```

The stoppe script is shown below:

```
#!/bin/ksh

su - db2pe -c db2stop
```

6.4.2 HACMP Rotating Standby

In the Rotating Standby configuration, Host 0 and Host 2 are both set up to be Node 0 of a DB2 PE instance, and Host 1 holds Node 1. The first machine to start out of Host 0 and Host 2 will become Node 0. If this machine fails, then the other machine will take over the resources for Node 0, and restart DB2 PE. The HACMP cluster configuration is shown below:

```
Cluster Description of Cluster db2pe
Cluster ID: 1
There were 2 networks defined : net1, net2
There are 2 nodes in this cluster.

NODE host0:
        This node has 2 service interface(s):

        Service Interface host0:
                IP address:     9.3.1.88
                Hardware Address:       0x42005a4f4165
                Network:        net1
                Attribute:      public

        Service Interface host0 has a possible boot configuration:
                Boot (Alternate Service) Interface: host0_boot
                IP address:     9.3.1.188
                Network:        net1
                Attribute:      public

        Service Interface host0 has 1 standby interfaces.
                Standby Interface 1: host0_standby
                IP address:     9.3.4.88
                Network:        net1
                Attribute:      public


        Service Interface host0_tty1:
                IP address:     /dev/tty1
                Hardware Address:
                Network:        net2
                Attribute:      serial

        Service Interface host0_tty1 has no standby interfaces


NODE host2:
        This node has 2 service interface(s):

        Service Interface host0:
                IP address:     9.3.1.88
                Hardware Address:       0x42005a4f4165
                Network:        net1
                Attribute:      public

        Service Interface host0 has a possible boot configuration:
                Boot (Alternate Service) Interface: host2_boot
                IP address:     9.3.1.178
                Network:        net1
                Attribute:      public

        Service Interface host0 has 1 standby interfaces.
                Standby Interface 1: host2_standby
                IP address:     9.3.4.78
                Network:        net1
                Attribute:      public


        Service Interface host2_tty0:
```

```
                IP address:     /dev/tty0
                Hardware Address:
                Network:        net2
                Attribute:      serial

        Service Interface host2_tty0 has no standby interfaces



Breakdown of network connections:

Connections to network net1
        Node host0 is connected to network net1 by these interfaces:
                host0_boot
                host0
                host0_standby

        Node host2 is connected to network net1 by these interfaces:
                host2_boot
                host0
                host2_standby


Connections to network net2
        Node host0 is connected to network net1 by these interfaces:
                host0_tty1

        Node host2 is connected to network net1 by these interfaces:
                host2_tty0
```

The resource configuration for this setup is shown below:

```
Resource Group Name               db2r
Node Relationship                 rotating
Participating Node Name(s)        host0 host2
Service IP Label                  host0
Filesystems                       /database /u/db2pe
                                  /usr/lpp/db2pe_01_02
Filesystems to be exported        /u/db2pe /usr/lpp/db2pe_01_02
Filesystems to be NFS mounted
Volume Groups
Concurrent Volume Groups
Disks
Application Servers               db2pe
Miscellaneous Data
Inactive Takeover                 false
9333 Disk Fencing                 false


Run Time Parameters:

Node Name                         host0
Debug Level                       high
Host uses NIS or Name Server      false

Node Name                         host2
Debug Level                       high
Host uses NIS or Name Server      false
```

An application server, db2pe, is defined to the cluster. This uses two scripts, startpe and stoppe, to control the DB2 PE environment. The startpe script is shown below:

```
#!/bin/ksh

su - db2pe -c db2start
```

The stoppe script is shown below:

```
#!/bin/ksh

su - db2pe -c db2stop
```

6.5 DRDA Application Server Feature

The DB2 Parallel Edition Distributed Relational Database Architecture (DRDA) Application Server (AS) feature (db2pe_01_02.drda) is supported in DB2 Parallel Edition V1.2. It enables DB2 Parallel Edition to function as a database server for Application Requestors using the DRDA protocol in addition to other clients that use DB2 private protocols (DB2/6000 Client Support). It provides support for DRDA Level 1.

For more information about DRDA concepts and a detailed description of DRDA commands and bind options, refer to the following publications: *Distributed Relational Database Architecture Reference, (SC26-4651-01) DDM Architecture Reference Manual Level 4, (SC21-9526-05)*

For more information about setting up a DRDA network, or connecting products on different platforms, refer to the following publication: *Distributed Relational Database Architecture Connectivity Guide, (SC26-4783-03)* For more information about DRDA AS in DB2 Parallel Edition, refer to *DB2 Parallel Edition for AIX Administration Guide and Reference* .

Because DRDA Application Servers and Application Requesters communicate using the APPC communication protocol, the SNA support option (db2pe_01_02.sna) must also be installed.

6.5.1 Supported DRDA Application Requesters

The following IBM ARs are supported:

- DB2 for MVS Version 2.3 with PTFs UN75958, and UN54600, and the fix for APAR PN83426.
- DB2 for MVS Version 3.1 with PTF UN75959, and the fix for APAR PN83426.
- DB2 for MVS Version 4.1 with the fix for APAR PN83426.
- DB2 for VM Version 3.3 with PTF UN47865
- DB2 for VM Version 3.4
- OS/400 Version 2.3.0 with PTFs SF23100, SF23205, SF23101, SF23722, SF23987, and SF23990

- OS/400 Version 3.0.5 with PTFs SF23950, SF23994, SF23986, SF23988, and SF23989
- OS/400 Version 3.1.0 with PTFs SF23270, SF23277, SF23271, SF23721, SF23985, and SF23960

6.5.2 SNA Server/6000 Customization

SNA Server/6000 profiles must be defined to provide APPC (LU 6.2) connectivity for the DB2 Parallel Edition DRDA AS feature. The AIX Systems Management Interface Tool (SMIT) is used to define and customize the profiles. All of the following profiles were created by starting with the SMIT Advanced SNA Configuration screen. This screen is found by selecting the following order of SMIT screens:

1. Communications Applications and Services
2. SNA Server/6000
3. Configure SNA Profiles
4. Advanced Configuration

6.5.2.1 SNA System Defaults

The DB2 Parallel Edition sysadm group must be defined as a trusted group to allow access to the following SNA functions:

- allocate with SECUR_SAME
- use of EXTRACT_FMH5 in the ioctl() or snactl() calls

Choose the subsequent SMIT screens in the following order to update the SNA System Defaults:

1. SNA System Defaults
2. Change/Show a Profile

The following values were used:

```
Trusted group names   system db2grp
```

The resulting profile is shown in Figure 88.

```
                  Change / Show SNA Node Profile
  Profile name                                        sna
  Maximum number of sessions (1-5000)                [200]
  Maximum number of conversations (1-5000)           [200]
  Restart Action                                      once
  Dynamic inbound partner LU definitions allowed?     yes
  NMVT action when no NMVT process                    reject
 *Trusted group names                                [system db2grp]
  APPC security sense codes                           specific
  Start SNMP subagent when SNA is started?            no
  Time out limited resource sessions?                 no
    If yes, time-out value (1-3600 seconds)          [15]
  Standard output file/device                        [/dev/console]
  Standard error file/device                         [/var/sna.stderr]

  Comments                                           []
```

Figure 88. SNA Node Profile

6.5.2.2 SNA Control Point Profile

The control point profile distinguishes this control point from other control points that may be connected to the network. The control point name, combined with the network name, form a unique identifier (the fully qualified name) for this control point. Each control point name, when qualified by the network name, should be unique across all networks.

The XID node ID consists of 8 arbitrary hexadecimal characters that the control point uses to identify itself as a physical unit (PU). By convention, the first three characters of 071 identify the local node as a RISC System/6000 workstation, and the last five characters are set to the device serial number.

If you are connecting to a VTAM host, the value in the control point name should match the CPNAME= parameter in the VTAM PU statement, the value in the network name field should match the NETID= parameter in the VTAM start statement, and the XID node ID field should match the concatenation of the IDBLK= and IDNUM= parameters in the VTAM PU statement.

Choose SMIT screens in the following order to update the SNA Control Point Profile:

1. Control Point
2. Change/Show a Profile

The following values were used:

XID node name	07150898
NETWORK name	USIBMSC
CONTROL POINT name	SC50898

The resulting profile is shown in Figure 89.

```
                Change / Show Control Point Profile
 Profile name                                        [node_cp]
*XID node ID                                         [07150898]
*Network name                                        [USIBMSC]
*Control Point (CP) name                             [SC50898]
 Control Point alias                                 []
 Control Point type                                   appn_end_node
 Maximum number of cached routing trees              [500]
 Maximum number of nodes in the TRS database         [500]
 Route addition resistance                           [128]

 Comments                                             []
```

Figure 89. SNA Control Point Profile

6.5.2.3 SNA DLC Profile

An SNA data link control (DLC) profile identifies the adapter device driver and adapter characteristics. The following example defines a Token Ring dynamic link station that will listen for incoming client connection requests.

Choose SMIT screens in the following order to define the Token Ring SNA DLC Profile:

1. Links
2. Token Ring
3. Token Ring SNA DLC
4. Add a Profile

The following values were used:

Profile name	tok.00001
Data link device name	tok0
Dynamic link stations supported?	yes
Effective capacity	15974400

The resulting profile is shown in Figure 90.

```
                    Add Token Ring DLC Profile

*Profile name                                        [tok0.00001]
*Data link device name                               [tok0]
 Force disconnect time-out (1-600 seconds)           [120]
 User defined maximum I-Field size?                   no
    If yes, Max. I-Field size (265-30729)            [30729]
 Max. num of active link stations (1-255)            [100]
    Number reserved for inbound activation           [0]
    Number reserved for outbound activation          [0]
 Transmit window count (1-127)                       [16]
 Dynamic window increment (1-127)                    [1]
 Retransmit count (1-30)                             [8]
 Receive window count (1-127)                        [1]
 Ring access priority                                 0
 Inactivity time-out (1-120 seconds)                 [48]
 Response time-out (1-40, 500 msec intervals)        [4]
 Acknowledge time out (1-40, 500 msec intervals)     [1]
 Local link name                                     []
 Local SAP address (02-fa)                           [04]
 Trace base listening link station?                   no
    If yes, trace format                              long
*Dynamic link stations supported?                     yes

 Link Recovery Parameters
    Retry interval (1-10000 seconds)                 [60]
    Retry limit (1-500 attempts)                     [20]

 Dynamic Link Activation Parameters
    Solicit SSCP sessions?                           yes
    CP-CP sessions supported?                        yes
    Partner required to support CP-CP sessions       no

 Dynamic Link TG COS Characteristics
*   Effective capacity                               [15974400]
    Cost per connect time                            [0]
    Cost per byte                                    [0]
    Security                                          nonsecure
    Propagation delay                                 lan
    User defined 1                                   [128]
    User defined 2                                   [128]
    User defined 3                                   [128]

 Comments                                             []
```

Figure 90. SNA Token Ring Data Link Control Profile

6.5.2.4 SNA Local LU Profile

The local LU profile defines an independent logical unit that will be used by DB2 Parallel Edition. This LU name must be unique with respect to all the other local LUs defined on this node. Moreover, the fully qualified local LU name must be unique across the network. The fully qualified LU name is generated by SNA Server/6000 by concatenating this LU name to the network name defined in the Control Point Profile.

Choose SMIT screens in the following order to update the Local LU Profile:

1. Sessions
2. LU 6.2
3. LU 6.2 Local LU
4. Add a Profile

The following values were used:

```
Profile name     SC50898I
Local LU name    SC50898I
```

The resulting profile is shown in Figure 91.

```
                    Add LU 6.2 Local LU Profile
*Profile name                                        [SC50898I]
*Local LU name                                       [SC50898I]
 Local LU alias                                      []
 Local LU is dependent?                               no
    If yes,
       Local LU address (1-255)                      []
       System services control point
            (SSCP) ID (*, 0-65535)                   [*]
       Link Station Profile name                     []
 Conversation Security Access List Profile name      []
 Recovery resource manager (RRM) enabled?             no

 Comments                                            []
```

Figure 91. SNA LU 6.2 Local LU Profile

6.5.2.5 SNA LU 6.2 Mode Profile

The LU 6.2 Mode Profile is used to specify parameters that can be used to tune throughput, availability, and system resource requirements for sessions that are established between the local LU and partner LU.

Choose SMIT screens in the following order to configure a LU 6.2 Mode Profile:

1. Sessions
2. LU 6.2
3. LU 6.2 Mode
4. Add a Profile

The following values were used:

Profile name	IBMRDB
Mode name	IBMRDB
Maximum number of sessions	30
Minimum contention winners	15
Minimum contention losers	15
Receive pacing window	8
Maximum RU size	4096
Minimum RU size	1024

The resulting profile is shown in Figure 92.

```
                    Add LU 6.2 Mode Profile
*Profile name                                          [IBMRDB]
*Mode name                                             [IBMRDB]
*Maximum number of sessions (1-5000)                   [30]
*Minimum contention winners (0-5000)                   [15]
*Minimum contention losers (0-5000)                    [15]
 Auto activate limit (0-500)                           [0]
 Upper bound for adaptive receive pacing window        [16]
*Receive pacing window (0-63)                          [8]
*Maximum RU size (128,...,32768: multiples of 32)      [4096]
*Minimum RU size (128,...,32768: multiples of 32)      [1024]
 Class of Service (COS) name                           [#CONNECT]

 Comments                                               []
```

Figure 92. SNA LU 6.2 Mode Profile

6.5.2.6 LU 6.2 TPN Profile

The LU 6.2 TPN Profile defines SNA and AIX characteristics associated with a target transaction program (TP) on the local node.

Choose SMIT screens in the following order to update the LU 6.2 Transaction Program Name Profile:

1. Sessions
2. LU 6.2
3. LU 6.2 Transaction Program Name (TPN)
4. Add a Profile

The following values were used:

Profile name	NYSERVER
Transaction program name (TPN)	NYSERVER
Conversation type	basic
Full path to TP executable	/u/db2pe/sqllib/bin/db2acntp
Multiple instances supported?	yes
User ID	802

Notes:

1. NYSERVER is an arbitrary name that may be replaced with a value that is appropriate for your environment. It must, however, match the LINKATTR value for this location in the DB2/MVS **SYSIBM.SYSLOCATIONS** table.
2. The value for User ID (802 in our example) is the numeric userid (uid) of the instance owner; you can find it with the AIX `id` command.

The resulting profile is shown in Figure 93 on page 252.

```
                    Add LU 6.2 TPN Profile

*Profile name                                      [NYSERVER]
*Transaction program name (TPN)                    [NYSERVER]
 Transaction program name (TPN) is in hexadecimal?  no
 PIP data?                                          no
    If yes, subfields (0-99)                       [0]
 Use command line parameters?                       no
 Command_line_parameters                           []
*Conversation type                                  basic
 Sync level                                         none/confirm
 Resource security level                            none
    If access, Resource Security Access List Prof. []
*Full path to TP executable                        [/u/db2pe/sqllib/bin/db2acntp]
 Multiple instances supported?                      yes
*User ID                                           [802]
 Server synonym name                               []
 Restart action                                     once
 Communication type                                 signals
    If IPC, Communication IPC queue key            [0]
 Time out Attaches?                                 yes
    If yes, time-out value (0-3600 seconds)        [60[
 Standard input file/device                        [/dev/console]
 Standard output file/device                       [/dev/console]
 Standard error file/device                        [/dev/console]

 Comments                                          []
```

Figure 93. SNA LU 6.2 TPN Profile

6.5.2.7 Verify and Commit SNA Profiles

New or changed configuration profiles are stored in a working configuration database. Before you can actually use these profiles, you must use the verifysna command to ensure the profiles are correct and internally consistent.

By specifying the update (-U) option with verifysna, you can also transfer the verified profiles to the committed configuration database, which is used by SNA Server/6000 when running on the local system.

You can use either of the following methods to verify configuration profiles:

- verifysna command
- Verify Configuration Profiles option on the SNA Server/6000 Advanced Configuration or Quick Configuration menu

6.5.3 Configuring the Database Manager

The DRDA Application Server is started along with the APPC client support feature when the tpname parameter is defined in the server's database manager configuration file, and the DB2COMM (defined in the $HOME/sqllib/db2profile file) is undefined, or set to null or APPC. The tpname parameter must be set to the Transaction Program Name (TPN) defined in the SNA Server/6000 profiles.

When you set the svcename and/or tpname database manager configuration parameters to support remote TCP/IP and/or APPC clients, db2start tries to start the connection and interrupt processes

that are required to support remote clients. Not all the nodes, however, are necessarily set up to support remote clients, and db2start will fail on these nodes. To prevent this problem, you can modify the db2profile script to disable the starting of the connection and interrupt processes for a particular node. For example, if only the f01n01 host is set up to support APPC clients, the db2profile script should contain the following:

```
if [$(hostname) = "f01n01" ]
   then export DB2COMM=APPC
   else export DB2COMM=NONE
fi
```

The drda_heap_sz parameter specifies the number of 4 KB memory pages to allocate for use by the DRDA Application Server. A DRDA heap is allocated each time a DRDA Application Requestor connects to a database, and is freed when the DRDA AR disconnects from the database. The default value should be used unless you receive an error code indicating that the DRDA heap is not large enough.

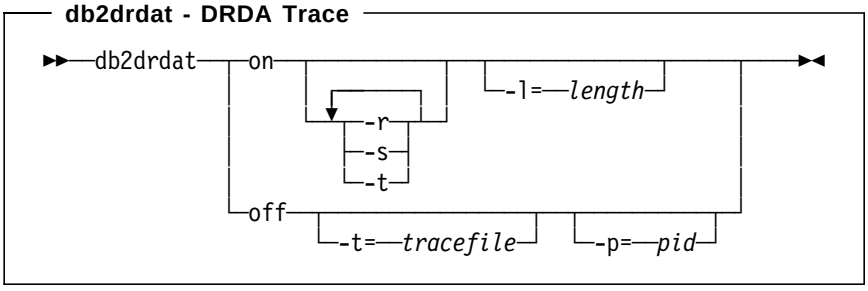
The following command will update the database manager configuration:

```
db2 update database manager configuration using tpname NYSERVER
```

6.5.4 Using DRDA Trace

The DRDA Trace command, db2drdat, can be used to capture the DRDA data stream exchanged between a DRDA Application Requestor (AR) and the DRDA Application Server (AS). This information is not only useful for problem determination, but also performance tuning in a client/server environment.

The syntax for db2drdat is as follows:



Appendix A. Special Notices

This publication is intended to help database administrators, system administrators, or anyone wanting to learn more about the parallel database environment. In particular, this publication covers DB2 Parallel Edition V1.2 for AIX, especially installation, configuration, and functions. See the PUBLICATIONS section of the IBM Programming Announcement for DB2 Parallel Edition V1.2 for AIX for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program, or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

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 of these techniques 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. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to

these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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 of these techniques 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. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

The following terms, which are denoted by an asterisk (*) in this publication, are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	AIX/6000
DATABASE 2	DB2
DB2/6000	Distributed Relational Database Architecture
DRDA	ES/9000
HACMP/6000	IBM
Micro Channel	POWERparallel
PROFS	RISC System/6000
RS/6000	SP2
400	9076 SP2

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Java and HotJava are trademarks of Sun Microsystems, Inc.

AFS	Transarc Corporation
ATM	Adobe Systems, Incorporated
NFS	Sun Microsystems, Incorporated
DEC, VAX	Digital Equipment Corporation
Interleaf	Interleaf, Incorporated

Other trademarks are trademarks of their respective companies.

Appendix B. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

B.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How To Get ITSO Redbooks” on page 261.

- *DB2/6000 Client/Server Usage Guide*, GG24-4322-00
- *Backup, Recovery and Availability with DB2 Parallel Edition on RS/6000 SP*, SG24-4695-00
- *Migrating and Managing Data on RS/6000 SP with DB2 Parallel Edition*, SG24-4658-00

B.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RISC System/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RISC System/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

B.3 Other Publications

These publications are also relevant as further information sources:

- *DB2 Parallel Edition for AIX, Administration Guide and Reference*, SC09-1982-01
- *DATABASE 2 AIX/6000 and DATABASE 2 OS/2 SQL Reference*, SC09-1574-00
- *DATABASE 2 AIX/6000 Command Reference*, SC09-1575-00
- *9076 Scalable POWERparallel 2:Administration Guide*, SH26-2486
- *Distributed Relational Database Architecture Connectivity Guide*, SC26-4783-03

How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

  http://w3.itso.ibm.com/redbooks

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269	
Canada	1-403-267-4455	
Outside North America	(+45) 48 14 2207	(long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 415 855 43 29 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number
- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

List of Abbreviations

AFS	Andrew File System
AIX	Advanced Interactive Executive
APPC	Advanced Program-to-Program Communication
AR	Application Requestor
AS	Application Server
ASC	Non-delimited ASCII
ASCII	American National Standard Code for Information Interchange
ATM	Asynchronous Transfer Mode
CLP	Command Line Processor
CLVM	Concurrent Logical Volume Manager
CPU	Central Processing Unit
CRM	Concurrent Resource Manager
CW	Control Workstation
DB2 PE	DATABASE 2 Parallel Edition for AIX
DDM	Distributed Data Management
DEC	Digital Equipment Corporation
DEL	Delimited ASCII
DDL	Data Definition Language
DFS	Distributed File System
DML	Data Manipulation Language
DRDA	Distributed Relational Database Architecture
DSS	Decision Support System
DPS	Data Protection Services
EBCDIC	Extended Binary Coded Decimal Interchange Code
FCM	Fast Communication Manager
FCS	Fibre Channel Standard
FDDI	Fiber Distributed Data Interface
FTP	File Transfer Program
HACMP	High Availability Cluster Multi-processing
HACWS	High Availability Control Workstation
HIPPI	High Performance Parallel Interface
HPS	High Performance Switch
IBM	International Business Machines Corporation
IP	Internet Protocol
ITSO	International Technical Support Organization
IXF	Integrated Exchange Format
LPP	Licensed Program Product
NFS	Network File System
NIS	Network Information Service
NTP	Network Time Protocol
ODBC	Open DataBase Connectivity
OLTP	Online Transaction Processing
PDB	Parallel Database
RAID	Redundant Array of Independent Disks
RDBMS	Relational Database Management System
SCSI	Small Computer Serial Interface
SMIT	System Management Interface Tool
SMP	Symmetric MultiProcessor
SOCC	Serial Optical Channel Converter
SQL	Structured Query Language
SSA	Serial Storage Architecture
TCP/IP	Transmission Control Protocol/Internet Protocol
VSD	Virtual Shared Disk
VTS	Virtual Timestamp
WSF	Work-Sheet Format

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

Printed in U.S.A.