CHAPTER 2

DB2 at a Glance: The Big Picture

This chapter is like a book within a book: it covers a vast range of topics that will provide you not only with a good introduction to DB2 core concepts and components, but also with an understanding of how these components work together and where they fit in the "DB2 puzzle." After reading this chapter you should have a general knowledge of the DB2 architecture that will help you better understand the topics discussed in the next chapters. Subsequent chapters will revisit and expand what has been discussed here.

In this chapter you will learn about:

- SQL statements and DB2 commands
- DB2 tools
- The DB2 environment
- Federation
- The database partitioning feature

You work with DB2 by issuing SQL statements and DB2 commands. To issue these statements and commands, you use DB2 tools. The DB2 tools interact with the DB2 environment by passing these statements and commands to the DB2 server for processing. This is shown in Figure 2.1.

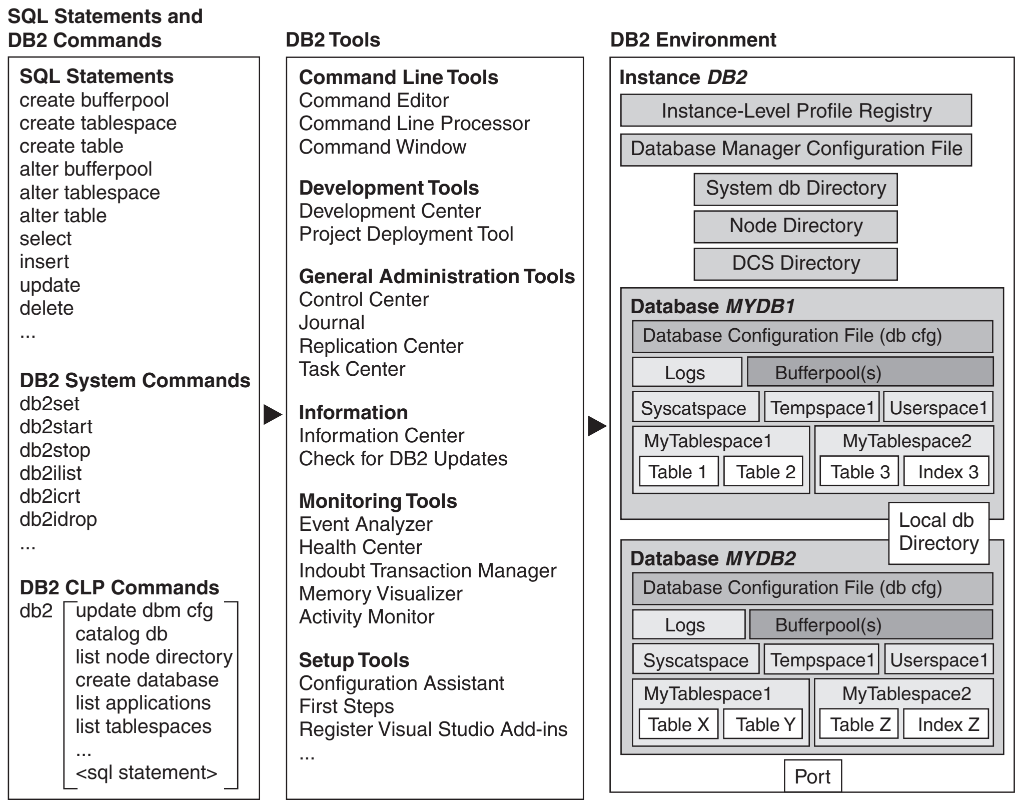


Figure 2.1 Overview of DB2

2.1 SQL Statements and DB2 Commands

SQL is the standard language used for retrieving and modifying data in a relational database. An SQL council formed by several industry leading companies determines the standard for these SQL statements, and the different relational database management systems (RDBMSs) follow these standards to make it easier for customers to use their databases. This section introduces the different categories of SQL statements and presents some examples.

DB2 commands are directives specific to DB2 that allow you to perform tasks against a DB2 server. There are two types of DB2 commands:

- System commands
- Command Line Processor (CLP) commands

NOTE SQL statements and DB2 commands can be specified in uppercase or lowercase. However, in Linux/UNIX some of the commands are case-sensitive; see Appendix B for a detailed explanation of the use of uppercase versus lowercase in DB2.

2.1.1 SQL Statements

SQL statements allow you to work with the data stored in your database. The statements are applied against the database you are connected to, not against the entire DB2 environment. There are three different classes of SQL statements.

- **Data Definition Language (DDL)** statements create, modify, or drop database objects. For example:
```
CREATE INDEX ix1 ON t1 (salary)
ALTER TABLE t1 ADD hiredate DATE
DROP VIEW view1
```
- **Data Manipulation Language (DML)** statements insert, update, delete, or select data from the database objects. For example:
```
INSERT INTO t1 VALUES (10,'Johnson','Peter')
UPDATE t1 SET lastname = 'Smith' WHERE firstname = 'Peter'
DELETE FROM t1
SELECT * FROM t1 WHERE salary > 45000
```
- **Data Control Language (DCL)** statements grant or revoke privileges or authorities to perform database operations on the objects in your database. For example:
```
GRANT  select ON employee TO peter
REVOKE update ON employee FROM paul
```

NOTE SQL statements are commonly referred to simply as "statements" in most RDBMS books. For detailed syntax of SQL statements, see the *DB2 UDB SQL Reference* manual.

NOTE The file *Command_and_SQL_Examples.pdf* on the CD-ROM accompanying this book includes a list of all SQL statements and DB2 commands and has examples for each one.

2.1.2 DB2 System Commands

You use DB2 system commands for many purposes, including starting services or processes, invoking utilities, and configuring parameters. Most DB2 system commands do not require the instance—the DB2 server engine process—to be started (instances are discussed later in this chapter). DB2 system command names have the format

```
db2x
```

where ***x*** represents one or more characters. For example:

```
db2start
db2set
db2icrt
```

NOTE Many DB2 system commands provide a quick way to obtain syntax and help information about the command by using the **`-h`** option. For example, typing **`db2set -h`** displays the syntax of the **`db2set`** command, with an explanation of its optional parameters.

2.1.3 DB2 Command Line Processor (CLP) Commands

DB2 CLP commands are processed by the CLP tool (introduced in the next section). These commands typically require the instance to be started, and they can be used for database and instance monitoring and for parameter configuration. For example:

```
list applications
create database
catalog tcpip node
```

You invoke the Command Line Processor by entering **`db2`** at an operating system prompt. If you enter **db2** and press the Enter key, you would be working with the CLP in interactive mode, and you can enter the CLP commands as shown above. On the other hand, if you don't want to work with the CLP in interactive mode, prefix each CLP command with **`db2`**. For example:

```
db2 list applications
db2 create database
db2 catalog tcpip node
```

Many books, including this one, display CLP commands as db2 *CLP_command* for this reason. Chapter 4, Using the DB2 Tools, explains the CLP in greater detail.

NOTE On the Windows platform, **`db2`** must be entered in the DB2 Command Window, not at the operating system prompt. The DB2 Command Window and the DB2 CLP are discussed in detail in Chapter 4, Using the DB2 Tools.

NOTE A quick way to obtain syntax and help information about a CLP command is to use the question mark (**`?`**) character followed by the command. For example:

```
db2 ? catalog tcpip node
```

or just

```
db2 ? catalog
```

For detailed syntax of a command, see the *DB2 UDB Command Reference* manual.

2.2 DB2 Tools Overview

Figure 2.2 shows all the tools available from the IBM DB2 menu. The IBM DB2 menu on a Windows system can be typically displayed by choosing **Start** > **Programs** > **IBM DB2**. On a Linux/UNIX system, the operating system's graphical support needs to be installed. DB2's graphical interface looks the same on all platforms. This section briefly introduces the tools presented in the IBM DB2 menu. Chapter 4 covers these tools in more detail, but for now simply familiarize yourself with them.

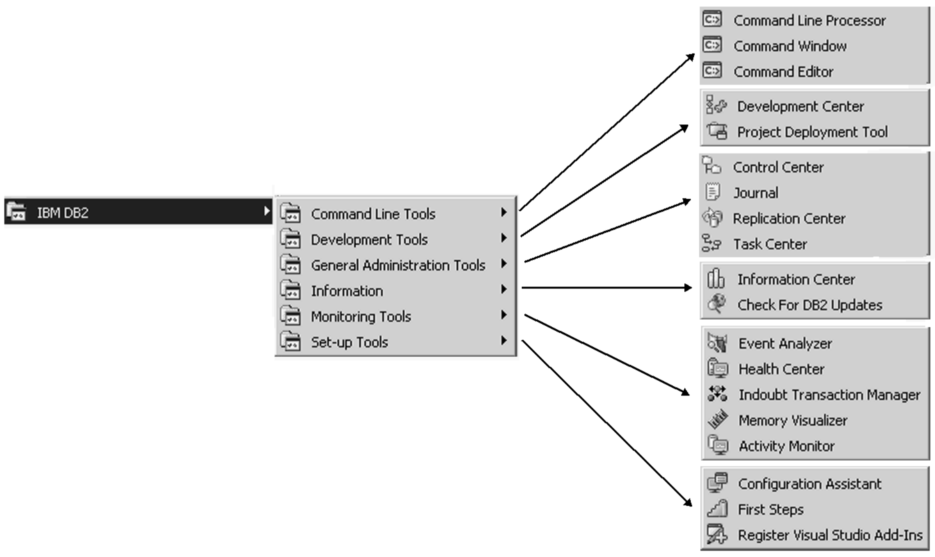


Figure 2.2 The IBM DB2 menus

2.2.1 Command Line Tools

Command line tools, as the name implies, allow you to issue DB2 commands and SQL statements from a command line interface. The two text-based interfaces are the Command Line Processor (CLP) and the Command Window. The Command Window is available only on Windows, while the CLP is available on all other platforms.

The Command Editor is a graphical interface tool that provides the same functionality as the text-based tools—and more. It also has the ability to invoke the Visual Explain tool, which shows the access path for the query.

2.2.2 Development Tools

Development tools allow developers to easily write and test stored procedures and user-defined functions, as well as to deploy them to different databases. DB2 provides two development tools.

- The **Development Center** provides developers with an environment where they can develop and test database application objects like stored procedures.
- The **Project Deployment Tool** lets developers deploy their Development Center project to another database.

2.2.3 General Administration Tools

The general administration tools allow database administrators (DBAs) to manage their database servers and databases from a central location.

- The **Control Center** is the most important of these tools. Not only does it support the administration of DB2 database servers on the Linux, UNIX, and Windows platforms, but also on the OS/390 and z/OS platforms. From the Control Center, database objects can be created, altered, and dropped. The tool also comes with several advisors to help you configure your system more quickly.
- The **Journal** tool can help investigate problems in the system. It tracks error messages and scheduled tasks that have been executed.
- The **Replication Center** lets you set up and manage your replication environment. Use DB2 replication when you want to propagate data from one location to another.
- The **Task Center** allows you to schedule tasks to be performed automatically. For example, you can arrange for a backup task to run at a time when there is minimal database activity.

2.2.4 Information Tools

The Information menu provides easy access to the DB2 documentation. The **Information Center** provides a fast and easy method to search the DB2 manuals. You can install the Information Center locally on your computer or intranet server, or access it via the Internet. Use the **Check for DB2 Updates** menu option to obtain the most up-to-date information about updates to the DB2 product.

2.2.5 Monitoring Tools

To maintain your database system, DB2 provides several tools that can help pinpoint the cause of a problem or even detect problems proactively before they cause performance deterioration.

- The **Event Analyzer** processes the information collected by an event monitor based on the occurrence of an event. For example, when two applications cannot continue their processing because the other is holding resources they need, a deadlock event occurs. This event is captured by an event monitor, and you can use the Event Analyzer to examine the captured data related to the deadlock and help resolve the contention. Some other events that can be captured are connections to the database, buffer pool activity, table space activity, table activity, SQL statements, and transactions.
- The **Health Center** detects problems before they happen by setting up thresholds which when exceeded cause alert notifications to be sent. The DBA can then choose to execute a recommended action to relieve the situation.

- The **Indoubt Transaction Manager** can help resolve issues with transactions that have been prepared but have not been committed or rolled back. This is only applicable to two-phase commit transactions.
- The **Memory Visualizer** tool lets you track the memory used by DB2. It plots a graph so you can easily monitor memory consumption.
- The **Activity Monitor** allows you to monitor application performance and concurrency, resource consumption, and SQL statement execution for a database. You can more easily diagnose problems with the reports this tool generates.

2.2.6 Setup Tools

The Setup tools help you configure your system to connect to remote servers, provide tutorials, and install add-ins to development tools.

- **First Steps** is a good starting point for new DB2 users who wish to become familiar with the product. This tool allows you to create a sample database and provides tutorials that help you familiarize yourself with DB2.
- The **Configuration Assistant** allows you to easily configure your system to connect to remote databases and to test the connection.
- The **Register Visual Studio Add-Ins** menu item lets you add a plug-in into Microsoft Visual Studio so that DB2 tools can be invoked from Visual Basic, Visual C++, and Visual InterDev. In each of these Microsoft development tools, the add-in inserts the DB2 menu entries into the tool's View, Tools, and Help menus. These add-ins provide Microsoft Visual Studio programmers with a rich set of application development tools to create stored procedures and user-defined functions designed for DB2.

2.2.7 Other Tools

The following are other DB2 tools that are not invoked directly from the DB2 menus.

- The **License Center** summarizes the licenses installed in your DB2 system and allows you to manage them.
- **Visual Explain** describes the access plan chosen by the DB2 optimizer, the brain of DB2, to access and retrieve information from tables.
- **SQL Assist** aids new users who are not familiar with the SQL language to write SQL queries.
- The **Satellite Administration Center** helps you set up and administer both satellites and the central satellite control server.

2.3 The DB2 Environment

Several items control the behavior of your database system. We first describe the DB2 environment on a single-partition database, and in section 2.6, Database Partitioning Feature, we expand the material to include concepts relevant to a multipartition database system (we don't want to overload you with information not required at this stage in the chapter).

Figure 2.3 provides an overview of the DB2 environment. Consider the following when you review this figure:

- The figure may look complex, but don't be overwhelmed by first impressions! Each item in the figure will be discussed in detail in the following sections.
- Since we reference Figure 2.3 throughout this chapter, *we strongly recommend that you bookmark page 31*. Alternatively, since this figure is available in color as a GIF file on the CD-ROM provided with this book (Figure_2_3.gif), consider printing it.
- The commands shown in the figure can be issued from the Command Window on Windows or the operating system prompt on Linux/UNIX. Chapter 4, Using the DB2 Tools, describes equivalent methods to perform these commands from the DB2 graphical tools.
- Each arrow points to a set of three commands. The first command in each set (in blue if you printed the figure using a color printer) inquires about the contents of a configuration file, the second command (in black) indicates the syntax to modify these contents, and the third command (in purple) illustrates how to use the command.
- The numbers in parentheses in Figure 2.3 match the superscripts in the headings in the following subsections.

2.3.1 An Instance$^{(1)}$

In DB2, an instance provides an independent environment where databases can be created and applications can be run against them. Because of these independent environments, databases in separate instances can have the same name. For example, in Figure 2.3 the database called *MYDB2* is associated to instance *DB2*, and another database called *MYDB2* is associated to instance *myinst*. Instances allow users to have separate, independent environments for production, test, and development purposes.

When DB2 is installed on the Windows platform, an instance named *DB2* is created by default. In the Linux and UNIX environments, if you choose to create the default instance, it is called *db2inst1*.

To create an instance explicitly, use:

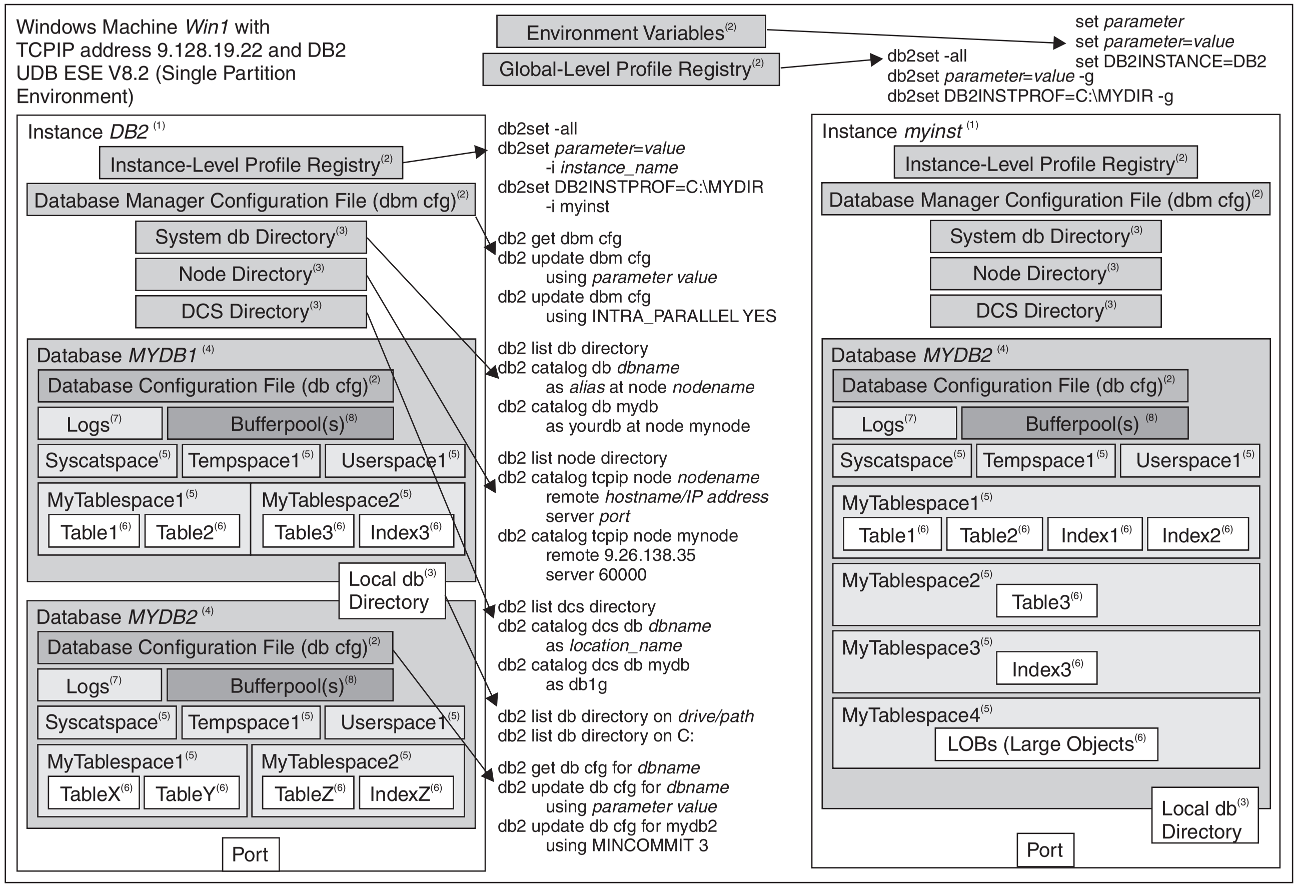
```
db2icrt instance_name
```

To drop an instance, use:

```
db2idrop instance_name
```

To start the current instance, use:

```
db2start
```



```
set parameter
set parameter=value
set DB2INSTANCE=DB2
```

```
db2set -all
db2set parameter=value -g
db2set DB2INSTPROF=C:\MYDIR -g
```

```
db2set -all
db2set parameter=value
        -i instance_name
db2set DB2INSTPROF=C:\MYDIR
        -i myinst

db2 get dbm cfg
db2 update dbm cfg
        using parameter value
db2 update dbm cfg
        using INTRA_PARALLEL YES

db2 list db directory
db2 catalog db dbname
        as alias at node nodename
db2 catalog db mydb
        as yourdb at node mynode

db2 list node directory
db2 catalog tcpip node nodename
        remote hostname/IP address
        server port
db2 catalog tcpip node mynode
        remote 9.26.138.35
        server 60000

db2 list dcs directory
db2 catalog dcs db dbname
        as location_name
db2 catalog dcs db mydb
        as db1g

db2 list db directory on drive/path
db2 list db directory on C:

db2 get db cfg for dbname
db2 update db cfg for dbname
        using parameter value
db2 update db cfg for mydb2
        using MINCOMMIT 3
```

Figure 2.3 The DB2 environment

To stop the current instance, use:

```
db2stop
```

When an instance is created on Linux and UNIX, logical links to the DB2 executable code are generated. For example, if the machine in Figure 2.3 was a Linux/UNIX machine and the instances *DB2* and *myinst* were created, both of them would be linked to the same DB2 code. A logical link works as an alias or pointer to another program. In Windows, there is a shared install path, and all instances access the same libraries and executables.

2.3.2 The Database Administration Server

The Database Administration Server (DAS) is a daemon or process running on the database server that allows for remote graphical administration from remote clients using the Control Center. If you don't need to administer your DB2 server using a graphical interface from a remote client, you don't need to start the DAS. There can only be one DAS per server machine regardless of the number of instances on the machine. Note that the DAS needs to be running at the database server you are planning to administer remotely, not at the DB2 client.

To start the DAS, use the command:

```
db2admin start
```

To stop the DAS, use the command:

```
db2admin stop
```

2.3.3 Configuration Files and the DB2 Profile Registries(2)

Like many other RDBMSs, DB2 uses different mechanisms to influence the behavior of the database management system. These include:

- Environment variables
- DB2 profile registry variables
- Configuration parameters

2.3.3.1 Environment Variables

Environment variables are defined at the operating system level. On Windows you can create a new entry for a variable or edit the value of an existing one by choosing **Control Panel** > **System** > **Advanced Tab** > **Environment Variables**. On Linux and UNIX you can normally add a line to execute the script `db2profile` (Bourne or Korn shell) or `db2cshrc` (C shell) (provided after DB2 installation), to the instance owner's .login or .profile initialization files.

The DB2INSTANCE environment variable allows you to specify the current active instance to which all commands apply. If DB2INSTANCE is set to *myinst*, then issuing the command `CREATE DATABASE mydb` will create a database associated to instance *myinst*. If you wanted to create this database in instance *DB2*, you would first change the value of the DB2INSTANCE variable to *DB2*.

Using the Control Panel (Windows) or the user profile (Linux/UNIX) to set the value of an environment variable guarantees that value the next time you open a window or session. If you only want to change this value temporarily while in a given window or session, you can use the operating system **`set`** command on Windows, or **`export`** on Linux/UNIX. The command

```
set DB2INSTANCE=DB2  (on Windows)
```

or

```
export DB2INSTANCE=DB2  (on Linux and UNIX)
```

sets the value of the DB2INSTANCE environment variable to *DB2*. A common mistake when using the command is to leave spaces before and/or after the equal sign (=)—no spaces should be entered.

To check the current setting of this variable, you can use any of these three commands:

```
echo %DB2INSTANCE%  (Windows only)
set DB2INSTANCE
db2 get instance
```

For a list of all available instances in your system, issue the following command:

```
db2ilist
```

2.3.3.2 The DB2 Profile Registry

The word "registry" always causes confusion when working with DB2 on Windows. The DB2 profile registry variables, or simply the DB2 registry variables, have no relation whatsoever with the Windows Registry variables. The DB2 registry variables provide a centralized location where some key variables influencing DB2's behavior reside.

NOTE Some of the DB2 registry variables are platform-specific.

The DB2 Profile Registry is divided into four categories.

- The DB2 instance-level profile registry
- The DB2 global-level profile registry
- The DB2 instance node-level profile registry
- The DB2 instance profile registry

The first two are the most common ones. The main difference between the global-level and the instance-level profile registries, as you can tell from their names, is the level to which the variables apply. Global-level profile registry variables apply to all instances on the server. As you can see from Figure 2.3, this registry has been drawn outside of the two instance boxes. Instance-level profile registry variables apply to a specific instance. You can see separate instance-level profile registry boxes inside each of the two instances in the figure.

To view the current DB2 registry variables, issue the following command from the CLP:

```
db2set -all
```

You may get output like this:

```
[i] DB2INSTPROF=C:\PROGRAM FILES\SQLLIB
[g] DB2SYSTEM=PRODSYS
```

As you may have already guessed, **`[i]`** indicates the variable has been defined at the instance level, while **`[g]`** indicates that it has been defined at the global level.

The following are a few other commands related to DB2 Registry variables.

To view all the registry variables that can be defined in DB2, use this command:

```
db2set -lr
```

To set the value of a specific variable (in this example, DB2INSTPROF) at the global level, use:

```
db2set DB2INSTPROF="C:\PROGRAM FILES\SQLLIB" -g
```

To set a variable at the instance level for instance *myinst*, use:

```
db2set DB2INSTPROF="C:\MY FILES\SQLLIB" -i myinst
```

Note that for the above commands, the same variable has been set at both levels: the global level and the instance level. When a registry variable is defined at different levels, DB2 will always choose the value at the lowest level, in this case the instance level.

For the **`db2set`** command, as with the **`set`** command discussed earlier, there are no spaces before or after the equal sign.

Some registry variables require you to stop and start the instance (**`db2stop`**/**`db2start`**) for the change to take effect. Other registry variables do not have this requirement. Refer to the *DB2 UDB Administration Guide: Performance* for a list of variables that have this requirement.

2.3.3.3 Configuration Parameters

Configuration parameters are defined at two different levels: the instance level and the database level. The variables at each level are different (not like DB2 registry variables, where the same variables can be defined at different levels).

At the instance level, variables are stored in the Database Manager Configuration file (dbm cfg). Changes to these variables affect all databases associated to this instance, which is why Figure 2.3 shows a Database Manager Configuration file box defined per instance and outside the databases.

To view the contents of the Database Manager Configuration file, issue the command:

```
db2 get dbm cfg
```

To update the value of a specific variable, use:

```
db2 update dbm cfg using parameter value
```

For example:

```
db2 update dbm cfg using INTRA_PARALLEL YES
```

With Version 8, many of the Database Manager Configuration parameters are now "configurable online," meaning the change is dynamic—you don't need to stop and start the instance. The file *ConfigurationParameters.pdf* included on the CD-ROM accompanying this book provides a short description of the Database Manager Configuration parameters and indicates whether they are configurable online.

At the database level, parameter values are stored in the Database Configuration file (db cfg). Changes to these parameters only affect the specific database. In Figure 2.3 you can see there is a Database Configuration file box inside each of the databases defined.

To view the contents of the Database Configuration file, issue the command:

```
db2 get db cfg for dbname
```

For example:

```
db2 get db cfg for mydb2
```

To update a value of a specific variable, use:

```
db2 update db cfg for dbname using parameter value
```

For example:

```
db2 update db cfg for mydb2 using MINCOMMIT 3
```

With Version 8 many of these parameters are configurable online, meaning that the change is dynamic, and you no longer need to disconnect all connections to the database for the change to take effect. The file *ConfigurationParameters.pdf* included on the book's CD-ROM provides a short description of the Database Configuration parameters and indicates whether they are configurable online.

2.3.4 Connectivity and DB2 Directories(3)

In DB2, directories are used to store connectivity information about databases and the servers on which they reside. There are four main directories, which are described in the following subsections. The corresponding commands to set up database and server connectivity are also included; however, many users find the Configuration Assistant graphical tool very convenient to set up database and server connectivity.

Chapter 6, Configuring Client and Server Connectivity, discusses all the commands and concepts described in this section in detail, including the Configuration Assistant.

2.3.4.1 System Database Directory

The system database directory (or system db directory) is the main "table of contents" that contains information about all the databases to which you can connect from your DB2 system. As you can see from Figure 2.3, the system db directory is stored at the instance level.

To list the contents of the system db directory, use the command:

```
db2 list db directory
```

Any entry from the output of this command containing the word *Indirect* indicates that the entry is for a local database, that is, a database that resides on the database server on which you are working. The entry also points to the local database directory indicated by the *Database drive* item (Windows) or *Local database directory* (Linux/UNIX).

Any entry containing the word *Remote* indicates that the entry is for a remote database—a database residing on a server other than the one on which you are currently working. The entry also points to the node directory entry indicated by the *Node name* item.

To enter information into the system database directory, use the **`catalog`** command:

```
db2 catalog db dbname as alias  at node nodename
```

For example:

```
db2 catalog db mydb    as yourdb at node mynode
```

The **`catalog`** commands are normally used only when adding information for remote databases. For local databases, a catalog entry is automatically created after creating the database with the **`CREATE DATABASE`** command.

2.3.4.2 Local Database Directory

The local database directory contains information about databases residing on the server where you are currently working. Figure 2.3 shows the local database directory overlapping the database box. This means that there will be one local database directory associated to all of the databases residing in the same location (the drive on Windows or the path on Linux/UNIX). The local database directory does not reside inside the database itself, but it does not reside at the instance level either; it is in a layer between these two. (After you read section 2.3.10, The Internal Implementation of the DB2 Environment, it will be easier to understand this concept.)

Note also from Figure 2.3 that there is no specific command used to enter information into this directory, only to retrieve it. When you create a database with the **`CREATE DATABASE`** command, an entry is added to this directory.

To list the contents of the local database directory, issue the command:

```
db2 list db directory on drive / path
```

where ***`drive`*** can be obtained from the item *Database drive* (Windows) or ***`path`*** from the item *Local database directory* (Linux/UNIX) in the corresponding entry of the system db directory.

2.3.4.3 Node Directory

The node directory stores all connectivity information for remote database servers. For example, if you use the TCP/IP protocol, this directory shows entries such as the host name or IP address of the server where the database to which you want to connect resides, and the port number of the associated DB2 instance.

To list the contents of the node directory, issue the command:

```
db2 list node directory
```

To enter information into the node directory, use:

```
db2 catalog tcpip node node_name
    remote hostname or IP_address
    server service_name or port_number
```

For example:

```
db2 catalog tcpip node mynode
    remote 192.168.1.100
    server 60000
```

You can obtain the port number of the remote instance to which you want to connect by looking at the SVCENAME parameter in the Database Manager Configuration file of that instance. If this parameter contains a string value rather than the port number, you need to look for the corresponding entry in the TCP/IP services file mapping this string to the port number.

2.3.4.4 Database Connection Services Directory

The Database Connection Services (DCS) directory contains connectivity information for host databases residing on a zSeries (z/OS or OS/390) or iSeries (OS/400) server. You need to have DB2 Connect software installed unless the server you are working on has DB2 UDB Enterprise Server Edition (ESE) installed. DB2 ESE comes with DB2 Connect support built in.

To list the contents of the DCS directory, issue the following command:

```
db2 list dcs directory
```

To enter information into the DCS directory, use:

```
db2 catalog dcs db dbname as location_name
```

For example:

```
db2 catalog dcs db mydb as db1g
```

2.3.5 Databases[(4)]

A database is a collection of information organized into interrelated objects like table spaces, tables, and indexes. Databases are closed and independent units associated to an instance. Because of this independence, objects in two or more databases can have the same name. For example, Figure 2.3 shows a table space called *MyTablespace1* inside the database *MYDB1* associated to instance *DB2*. Another table space with the name *MyTablespace1* is also used inside the database *MYDB2*, which is also associated to instance *DB2*.

Since databases are closed units, you cannot perform queries involving tables of two different databases in a direct way. For example, a query involving *Table1* in database *MYDB1* and *TableZ* in database *MYDB2* is not readily allowed. For an SQL statement to work against tables of different databases, you need to use *federation* (see section 2.4, Federation).

You create a database with the command **`CREATE DATABASE`**. This command automatically creates three table spaces, a buffer pool, and several configuration files, which is why this command can take a few seconds to complete.

NOTE While **`CREATE DATABASE`** looks like an SQL statement, it is considered a DB2 CLP command.

2.3.6 Table Spaces(5)

Table spaces are logical objects used as a layer between logical tables and physical containers. **Containers** are where the data is physically stored in files, directories, or raw devices. When you create a table space, you can associate it to a specific buffer pool (database cache) and to specific containers.

Three table spaces—the catalog (SYSCATSPACE), system temporary space (TEMPSPACE1), and the default user table space (USERSPACE1)—are automatically created when you create a database. The catalog and the system temporary space can be considered system structures, as they are needed for the normal operation of your database. The catalog contains **metadata** (data about your database objects) and must exist at all times. Some other RDBMSs call this structure a "data dictionary."

NOTE Do not confuse the term "catalog" in this section with the **`catalog`** command mentioned earlier; they have no relationship at all.

A system temporary table space is the work area for the database manager to perform operations, like joins and overflowed sorts. There must be at least one system temporary table space in each database.

The USERSPACE1 table space is created by default, but you can delete it. To create a table in a given table space, use the **`CREATE TABLE`** statement with the **`IN`** ***`table_space_name`*** clause. If a table space is not specified in this statement, the table will be created in the first user-created table space. If you have not yet created a table space, the table will be created in the USERSPACE1 table space.

Figure 2.3 shows other table spaces that were explicitly created with the **`CREATE TABLESPACE`** statement (in brown in the figure on the CD-ROM). Chapter 8, The DB2 Storage Model, discusses table spaces in more detail.

2.3.7 Tables, Indexes, and Large Objects$^{(6)}$

A **table** is an unordered set of data records consisting of columns and rows. An **index** is an ordered set of pointers associated with a table, and is used for performance purposes and to ensure uniqueness. Nontraditional relational data, such as video, audio, and scanned documents, are stored in tables as large objects (LOBs). Tables and indexes reside in table spaces. Chapter 8 describes these in more detail.

2.3.8 Logs$^{(7)}$

Logs are used by DB2 to record every operation against a database. In case of a failure, logs are crucial to recover the database to a consistent point. See Chapter 13, Developing Backup and Recovery Solutions, for more information about logs.

2.3.9 Buffer Pools$^{(8)}$

A **buffer pool** is an area in memory where all index and data pages other than LOBs are processed. DB2 retrieves LOBs directly from disk. Buffer pools are one of the most important objects to tune for database performance. Chapter 8, The DB2 Storage Model, discusses buffer pools in more detail.

2.3.10 The Internal Implementation of the DB2 Environment

We have already discussed DB2 registry variables, configuration files, and instances. In this section we illustrate how some of these concepts physically map to directories and files in the Windows environment. The structure is a bit different in Linux and UNIX environments, but the main ideas are the same. Figures 2.4, 2.5, and 2.6 illustrate the DB2 environment internal implementation that corresponds to Figure 2.3.

Figure 2.4 shows the directory where DB2 was installed: H:\Program Files\IBM\SQLLIB. The SQLLIB directory contains several subdirectories and files that belong to DB2, including the binary code that makes DB2 work, and a subdirectory is created for each instance that is created on the machine. For example, in Figure 2.4 the subdirectories DB2 and MYINST correspond to the instances *DB2* and *myinst* respectively. The DB2DAS00 subdirectory corresponds to the DAS.

At the top of the figure there is a directory H:\MYINST. This directory contains all the databases created under the H: drive for instance *myinst*. Similarly, the H:\DB2 directory contains all the databases created under the H: drive for instance *DB2*.

Figure 2.5 shows an expanded view of the H:\Program Files\IBM\SQLLIB\DB2 directory. This directory contains information about the instance *DB2*. The db2systm binary file contains the database manager configuration (dbm cfg). The other two files highlighted in the figure (db2nodes.cfg and db2diag.log) are discussed later in this book. For now, the description of these files in the figure is sufficient. The figure also points out the directories where the system database, Node, and DCS directories reside. Note that the Node and DCS directories don't exist if they don't have any entries.

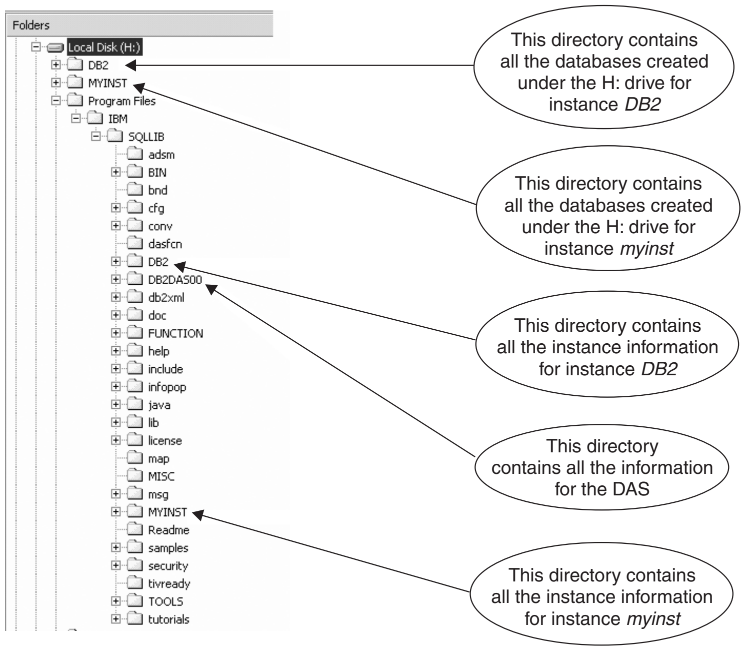

Folders
Local Disk (H:)
DB2
MYINST
Program Files
IBM
SQLLIB
adsm
BIN
bnd
cfg
conv
dasfcn
DB2
DB2DAS00
db2xml
doc
FUNCTION
help
include
infopop
java
lib
license
map
MISC
msg
MYINST
Readme
samples
security
tivready
TOOLS
tutorials
This directory contains all the databases created under the H: drive for instance *DB2*
This directory contains all the databases created under the H: drive for instance *myinst*
This directory contains all the instance information for instance *DB2*
This directory contains all the information for the DAS
This directory contains all the instance information for instance *myinst*

Figure 2.4 The internal implementation environment for DB2 for Windows

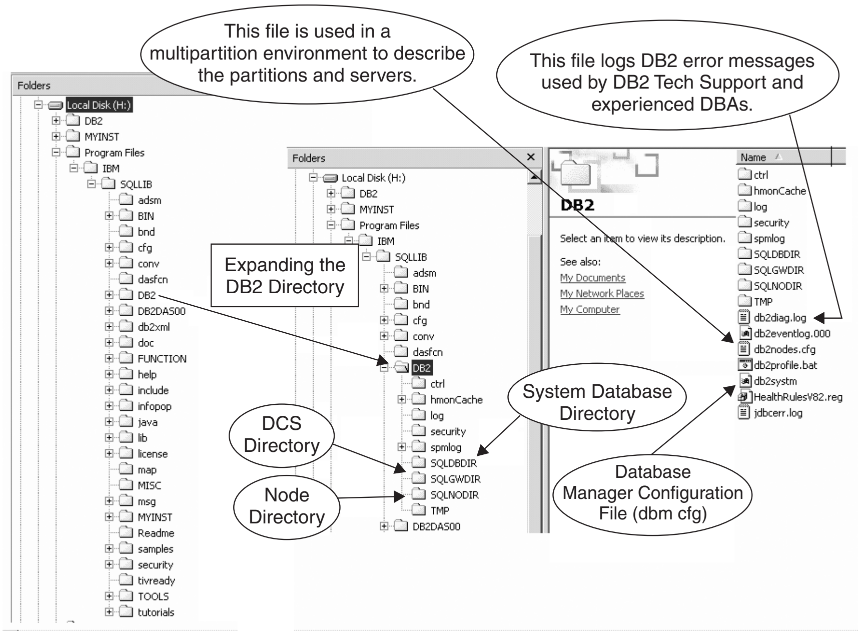


Figure 2.5 Expanding the DB2 instance directory

In Figure 2.6, the H:\DB2 and H:\MYINST directories have been expanded. The subdirectories SQL00001 and SQL00002 under H:\DB2\NODE0000 correspond to the two databases created under instance *DB2*. To map these directory names to the actual database names, you can review the contents of the local database directory with this command:

```
list db directory on h:
```

Chapter 6, Configuring Client and Server Connectivity, shows sample output of this command. Note that the local database directory is stored in the subdirectory SQLDBDIR. This subdirectory is at the same level as each of the database subdirectories; therefore, when a database is dropped, this subdirectory is not dropped. Figure 2.6 shows two SQLDBDIR subdirectories, one under H:\DB2\NODE0000 and another one under H:\MYINST\NODE0000.

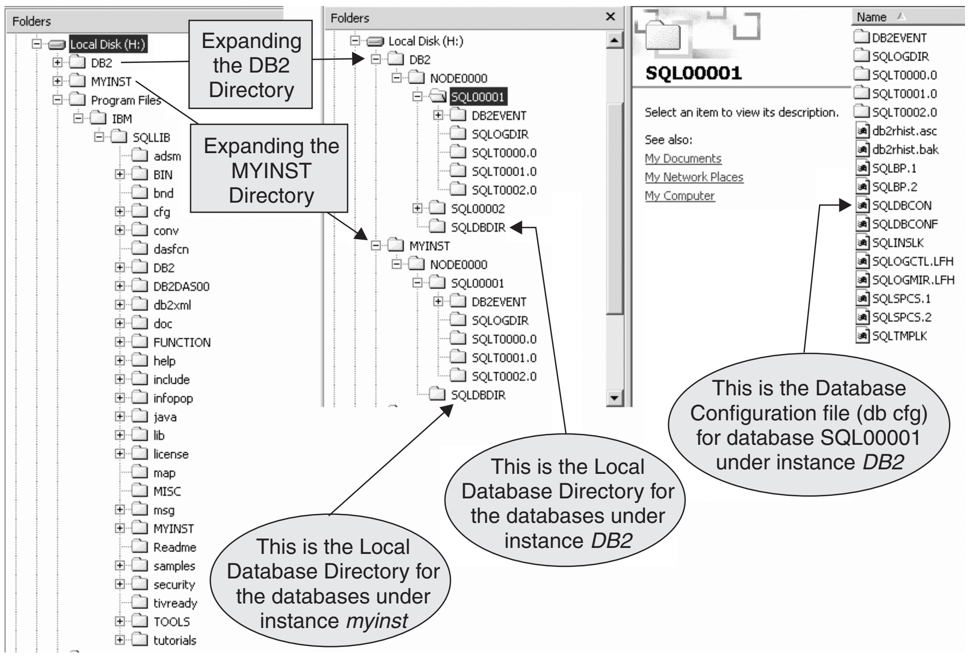


Figure 2.6 Expanding the directories containing the database data

Knowing how the DB2 environment is internally implemented can help you understand the DB2 concepts better. For example, looking back at Figure 2.3 (that one you should have printed!), what would happen if you dropped the instance *DB2*? Would this mean that databases *MYDB1* and *MYDB2* are also dropped? The answer is no. Figure 2.4 clearly shows that the directory where the instance information resides (H:\Program Files\IBM\SQLLIB\DB2) and the directory where the data resides (H:\DB2) are totally different. When an instance is dropped, only the subdirectory created for that instance is dropped.

Similarly, let's say you uninstall DB2 at a given time, and later you reinstall it on the same drive. After reinstallation, can you access the "old" databases created before you uninstalled DB2 the first time? The answer is yes. When you uninstalled DB2, you removed the SQLLIB directory, therefore the DB2 binary code as well as the instance subdirectories were removed, but the databases were left untouched. When you reinstall DB2, a new SQLLIB directory is created with a new default DB2 instance; no other instance is created. The new DB2 instance will have a new empty system database directory (db2systm). So even though the directories containing the database data were left intact, you need to explicitly put the information in the DB2 system database directory for DB2 to recognize the existence of these databases. For example, if you would like to access the MYDB1 database of the *DB2* instance, you need to issue this command to add an entry to the system database directory:

```
catalog db mydb1 on h:
```

If the database you want to access is MYDB2 that was in the *myinst* instance, you would first need to create this instance, switch to the instance, and then issue the **`catalog`** command as shown below.

```
db2icrt myinst
set DB2INSTANCE=myinst
catalog db mydb2 on h:
```

It is a good practice to back up the contents of all your configuration files as shown below.

```
db2 get dbm cfg > dbmcfg.bk
db2set -all > db2set.bk
db2 list db directory > systemdbdir.bk
db2 list node directory > nodedir.bk
db2 list dcs directory > dcsdir.bk
```

Notice that all of these commands redirect the output to a text file with a .bk extension.

CAUTION The purpose of this section is to help you understand the DB2 environment by describing its internal implementation. We strongly suggest that you *do not tamper with the files and directories discussed in this section*. You should only modify the files using the commands described in earlier sections.

2.4 Federation

Database federated support in DB2 allows tables from multiple databases to be presented as local tables to a DB2 server. The databases may be local or remote; they can also belong to different RDBMSs. While Chapter 1 briefly introduced federated support, this section provides an overview of how federation is implemented.

First of all, make sure that your server allows federated support: The database manager parameter FEDERATED must be set to YES.

DB2 uses NICKNAME, SERVER, WRAPPER, and USER MAPPING objects to implement federation. Let's consider the example illustrated in Figure 2.7.

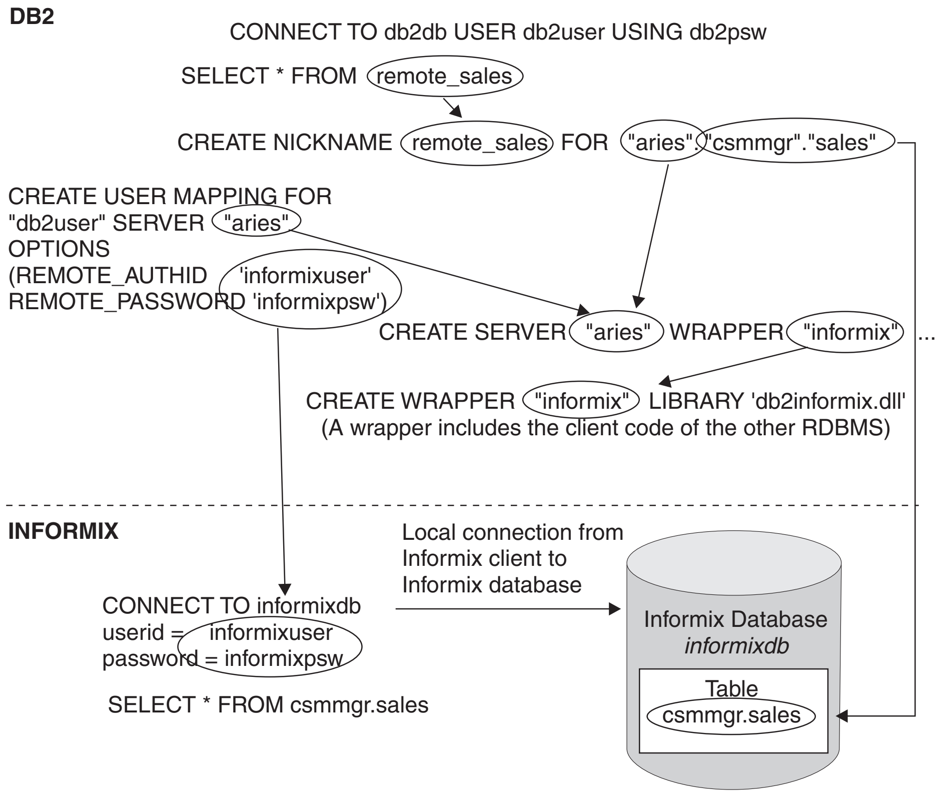

DB2
CONNECT TO db2db USER db2user USING db2psw
SELECT * FROM remote_sales
CREATE NICKNAME remote_sales FOR "aries"."csmmgr"."sales"
CREATE USER MAPPING FOR "db2user" SERVER "aries" OPTIONS (REMOTE_AUTHID 'informixuser' REMOTE_PASSWORD 'informixpsw')
CREATE SERVER "aries" WRAPPER "informix" ...
CREATE WRAPPER "informix" LIBRARY 'db2informix.dll'
(A wrapper includes the client code of the other RDBMS)
INFORMIX
Local connection from Informix client to Informix database
CONNECT TO informixdb userid = informixuser password = informixpsw
SELECT * FROM csmmgr.sales
Informix Database
informixdb
Table
csmmgr.sales

Figure 2.7 An overview of a federation environment

The DB2 user *db2user* connects to the database *db2db*. He then issues the statement:

```
SELECT * FROM remote_sales
```

The table ***remote_sales***, however, is not a local table but a **nickname**, which is a pointer to a table in another database, possibly in another server and from a different RDBMS. A nickname is created with the **CREATE NICKNAME** statement, and requires a SERVER object (*aries* in the example) and the schema and table name to be accessed at this server (*csmmgr.sales*).

A SERVER object is associated to a WRAPPER. A wrapper is associated to a library that contains all the code required to connect to a given RDBMS. For IBM databases like Informix, these wrappers or libraries are provided with DB2. For other RDBMSs, you need to obtain the IBM DB2 Information Integrator software. In Figure 2.7, the wrapper called *informix* was created, and it is associated to the library *db2informix.dll*.

To access the Informix table *csmmgr.sales*, however, you cannot use the DB2 user id and password directly. You need to establish a mapping between the DB2 user id and an Informix user id that has the authority to access the desired table. This is achieved with the **CREATE USER MAPPING** statement. Figure 2.7 shows how the DB2 user *db2user* and the Informix user *informixuser* are associated with this statement.

2.5 Case Study: The DB2 Environment

NOTE Several assumptions have been made in this case study and the rest of the case studies in this book, so if you try to follow them some steps may not work for you. If you do follow some or all of the steps in the case studies, we recommend you use a test computer system.

You recently attended a DB2 training class and would like to try things out on your own laptop at the office. Your laptop is running Windows 2000 and DB2 UDB Enterprise Server Edition has been installed. You open the Command Window and take the following steps.

1. First, you want to know how many instances you have in your computer, so you enter:
   ```
   db2ilist
   ```
2. Then, to find out which of these instances is the current active one, you enter:
   ```
   db2 get instance
   ```
   With the **`db2ilist`** command, you found out there were two instances defined on this computer, *DB2* and *myinst*. With the **`db2 get instance`** command, you learned that the *DB2* instance is the current active instance.
3. You would now like to list the databases in the *myinst* instance. Since this one is not the current active instance, you first switch to this instance temporarily in the current Command Window:
   ```
   set DB2INSTANCE=myinst
   ```
4. You again issue **`db2 get instance`** to check that *myinst* is now the current instance.
5. To list the databases defined on this instance you issue:
   ```
   db2 list db directory
   ```
   This command shows that you only have one database (*MYDB2*) in this instance.
6. You want to try creating a new database called *TEMPORAL*, so you execute:
   ```
   db2 create database temporal
   ```
   The creation of the database takes some time because several objects are created by default inside the database. Issuing another **`list db directory`** command now shows two databases: *MYDB2* and *TEMPORAL*.
7. You connect to the *MYDB2* database (**`db2 connect to mydb2`**) and check which tables you have in this database (**`db2 list tables for all`**). You also check how many table spaces are defined (**`db2 list tablespaces`**).
8. Next, you want to review the contents of the database configuration file (db cfg) for the *MYDB2* database:
   ```
   db2 get db cfg for mydb2
   ```

9. To review the contents of the Database Manager Configuration file (dbm cfg) you issue:

```
db2 get dbm cfg
```

10. At this point, you want to practice changing the value of a dbm cfg parameter, so you pick the INTRA_PARALLEL parameter which has a value set to YES. You change its value to NO as follows:

```
db2 update dbm cfg using INTRA_PARALLEL NO
```

11. You learned at the class that this parameter is not "configurable online," so you know you have to stop and start the instance. Since there is a connection to a database in the current instance (remember you connected to the *MYDB2* database earlier from your current Command Window), DB2 will not allow you to stop the instance. Enter the following sequence of commands:

```
db2 terminate (terminates the connection)
db2stop
db2start
```

And that's it! In this case study you have reviewed some basic instance commands like **`db2ilist`** and **`get instance`**. You have also reviewed how to switch to another instance, create and connect to a database, list the databases in the instance, review the contents of the database configuration file and the database manager configuration file, update a database manager configuration file parameter, and stop and start an instance.

2.6 Database Partitioning Feature

In this section we introduce you to the database partitioning feature (DPF) available on DB2 UDB Enterprise Server Edition (ESE). DPF lets you partition your database across multiple servers or within a large SMP server. This allows for scalability, since you can add new machines and spread your database across them. That means more CPUs, more memory, and more disks from each of the additional machines for your database!

DB2 UDB ESE with DPF is ideal to manage large databases, whether you are doing data warehousing, data mining, online analytical processing (OLAP), or working with online transaction processing (OLTP) workloads. You do not have to install any new code to enable this feature, but you must purchase the license before enabling the database partitioning feature. Users connect to the database and issue queries as usual without the need to know the database is spread among several partitions.

Up to this point, we have been discussing a single partition environment, and all of those concepts apply to a multipartition environment as well. We will now point out some implementation differences and will introduce a few new concepts, including database partitions, partition groups, and the coordinator partition, that are relevant only to a multipartition environment.

NOTE Prior to Version 8, DB2 UDB ESE with DPF was known as DB2 UDB Enterprise-Extended Edition (EEE).

2.6.1 Database Partitions

A **database partition** is an independent part of a partitioned database with its own data, configuration files, indexes, and transaction logs. You can assign multiple partitions across several physical servers or to a single physical server. In the latter case, the partitions are called **logical partitions** and they can share the machine's resources.

A single-partition database is a database with only one partition. We described the DB2 environment for this type of database in section 2.3, The DB2 Environment. A **multipartition database** (also referred to as a **partitioned database**) is a database with two or more database partitions. Depending on your hardware environment, there are several topologies for database partitioning. Figure 2.8 shows configurations of physical partitions, one partition per machine. The illustration at the top of the figure shows an SMP machine with one partition (single-partition environment). This means the entire database resides on this one machine. The illustration at the bottom

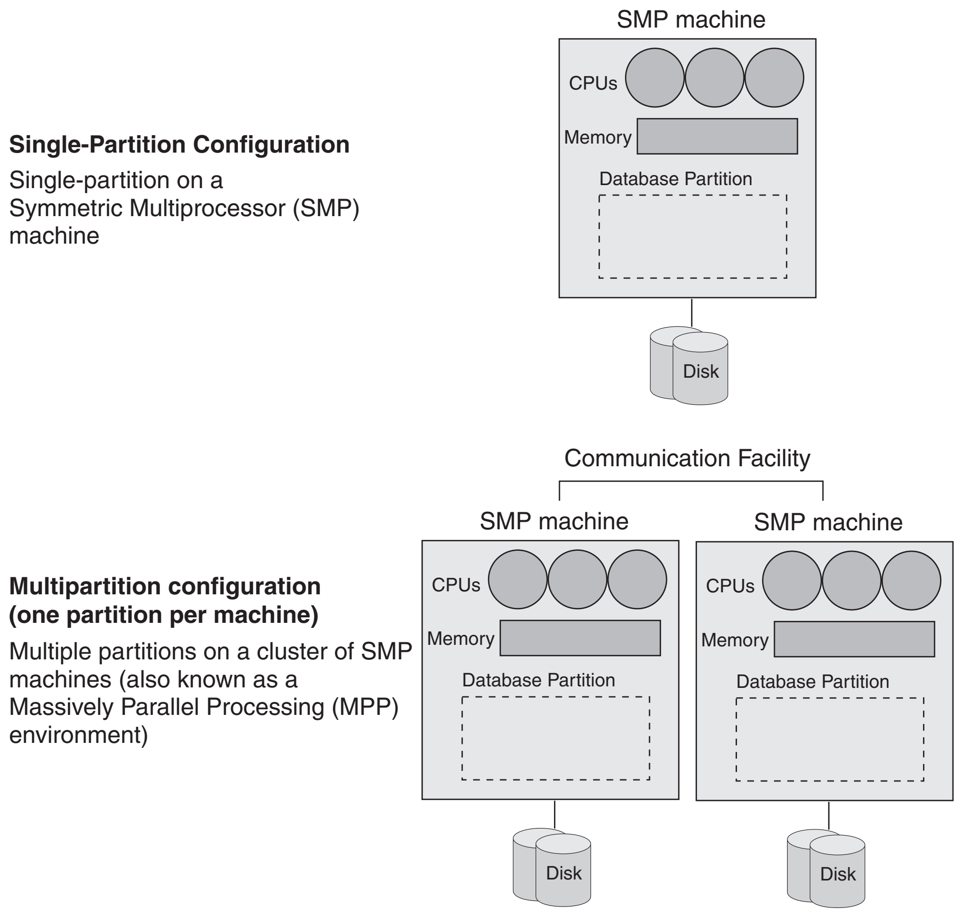

SMP machine
CPUs
Memory
Database Partition
Disk
Single-Partition Configuration
Single-partition on a Symmetric Multiprocessor (SMP) machine
Communication Facility
SMP machine
SMP machine
CPUs
Memory
Database Partition
Disk
Multipartition configuration (one partition per machine)
Multiple partitions on a cluster of SMP machines (also known as a Massively Parallel Processing (MPP) environment)

Figure 2.8 Database partition configurations with one partition per machine

shows two SMP machines, one partition per machine (multipartition environment). This means the database is split between the two partitions.

> **NOTE** In Figure 2.8, the symmetric multiprocessor (SMP) systems could be replaced by uniprocessor systems.

Figure 2.9 shows multipartition configurations with multiple partitions per machine. Unlike Figure 2.8 where there was only one partition per machine, this figure illustrates two (or more) partitions per machine.

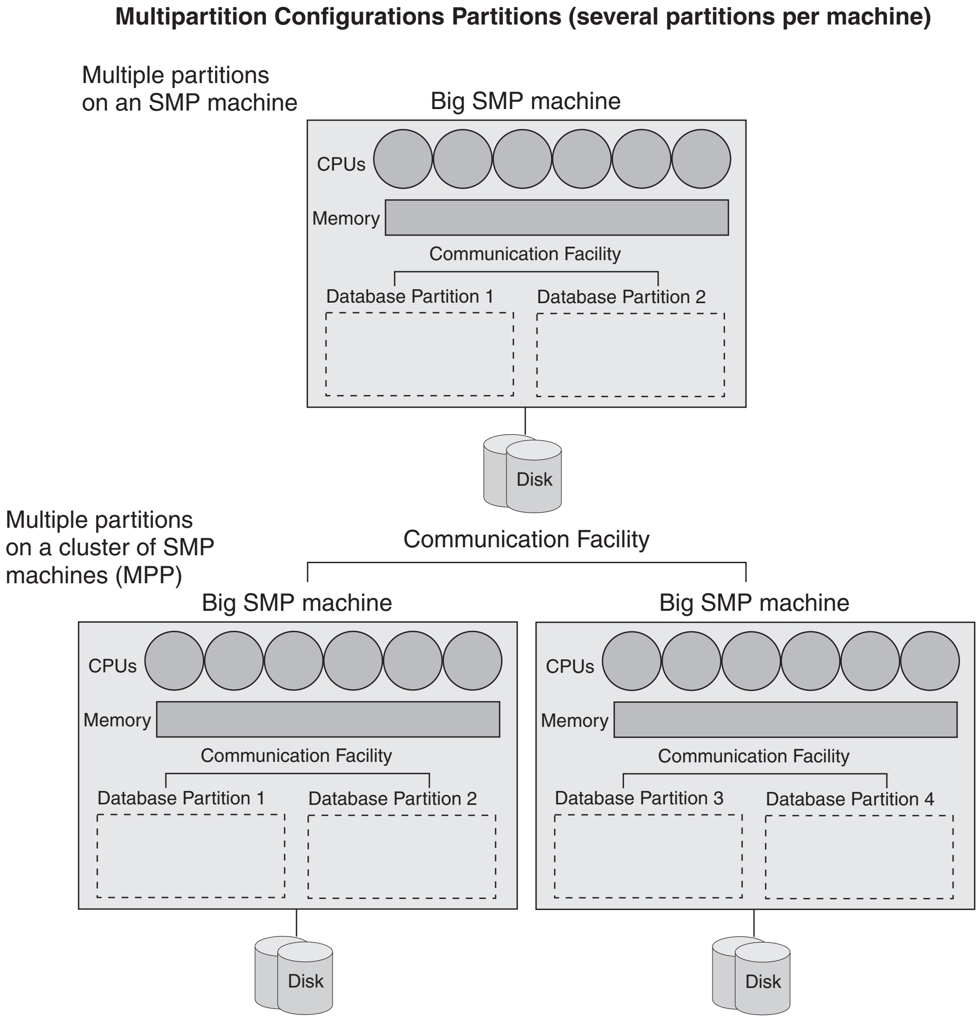


Figure 2.9 Database partition configurations with multiple partitions per machine

NOTE Prior to Version 8, the term "node" was used instead of "database partition." In Version 8, some commands will accept this term for compatibility with scripts written in previous versions of DB2.

Also, note that the *node directory* concept described in section 2.3.4.3, Node Directory, has no relationship whatsoever to the *database partition* concept, even though the term "node" is used.

To visualize how a DB2 environment is split in a DPF system, Figure 2.10 illustrates a partial reproduction of Figure 2.3, and shows it split into three physical partitions, one partition per server. (We have changed the machine in the original Figure 2.3 to use the Linux operating system instead of the Windows operating system.)

NOTE Since we reference Figure 2.10 throughout this section, *we recommend that you bookmark page 49*. Alternatively, since this figure is available in color on the CD-ROM provided with this book (Figure_2_10.gif), consider printing it.

In Figure 2.10, the DB2 environment is "split" so that it now resides on three servers running the same operating system (Linux, in this example). The partitions are also running the same DB2 version, but it is important to note that different FixPak levels are allowed. This figure shows where files and objects would be located on a new installation of a multipartition system.

It is also important to note that all of the machines participating in a DPF environment have to be interconnected by a high-speed communication facility that supports the TCP/IP protocol. TCP/IP ports are reserved on each machine for this "interpartition" communication. For example, by default after installation, the services file on Linux (/etc/services) is updated as follows (assuming you chose to create the *db2inst1* instance):

```
DB2_db2inst1          60000/tcp
DB2_db2inst1_1        60001/tcp
DB2_db2inst1_2        60002/tcp
DB2_db2inst1_END      60003/tcp
db2c_db2inst1         50000/tcp
```

This also depends on the number of partitions on the server. By default, ports 60000 through 60003 are reserved for interpartition communication. You can update the services file with the correct number of entries to support the number of partitions you are configuring.

When the partitions reside on the same machine, communication between the partitions still requires this setup. You force interpartition communication to be performed in memory by setting the DB2 registry variable DB2_FORCE_FCM_BP to YES.

For a DB2 client to connect to a DPF system, you issue **`catalog`** commands at the client to populate the system and node directories. In the example, the port number to use in these commands is 50000 to connect to the *db2inst1* instance, and the host name can be any of the servers

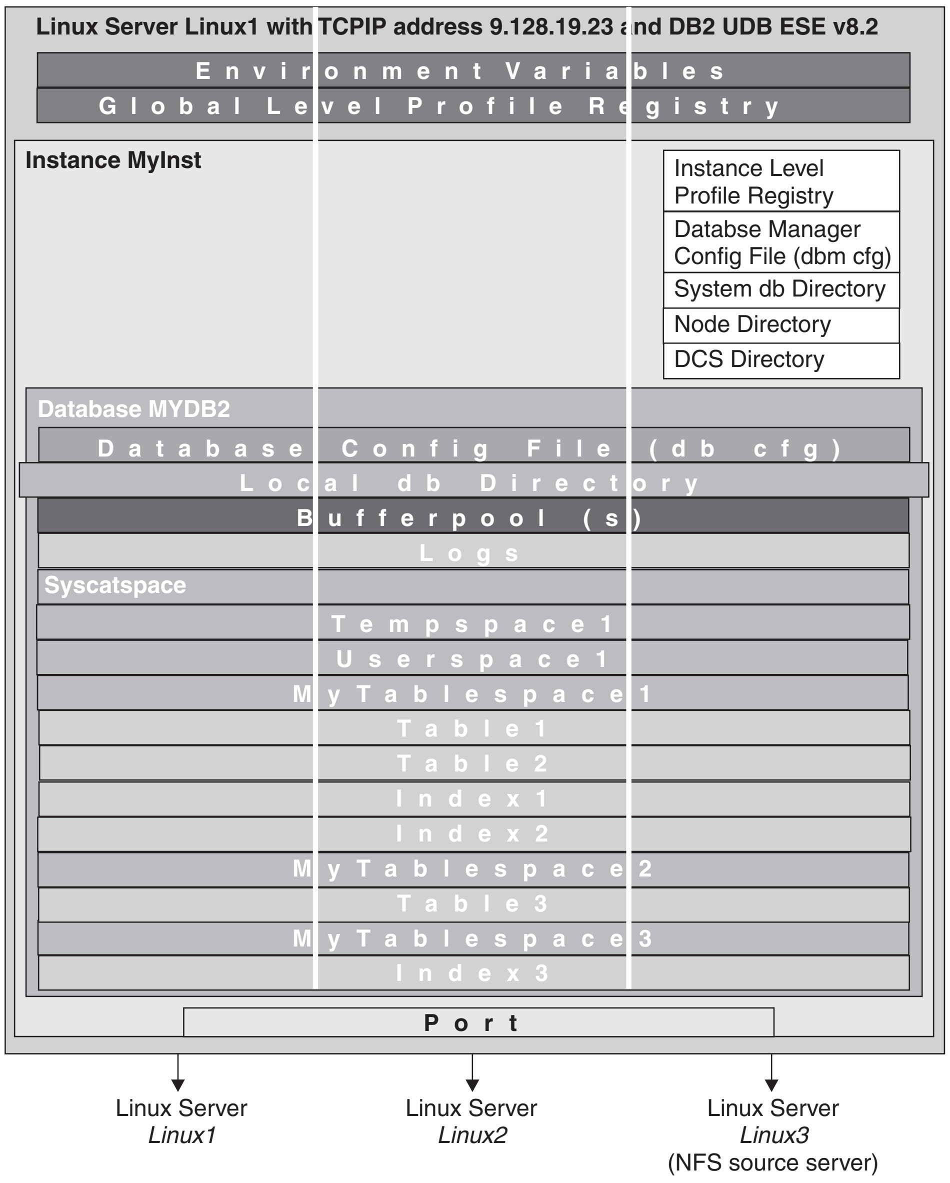


Figure 2.10 The DB2 environment in DB2 UDB ESE with DPF

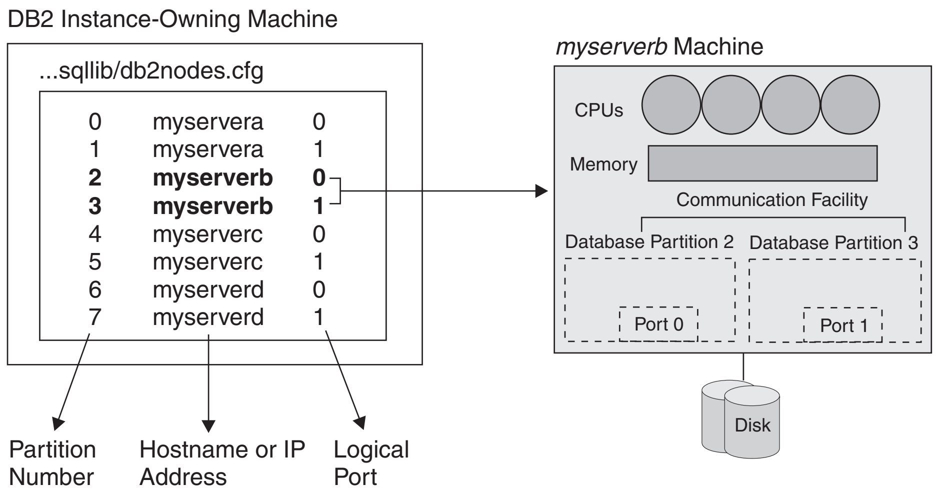
participating in the DPF environment. The server used in the `catalog` command becomes the coordinator, unless the DBPARTITIONNUM option of the `connect` statement is used. The concept of coordinator is described later in this section. Chapter 6, Configuring Client and Server Connectivity, discusses the `catalog` command in detail.

NOTE Each of the servers participating in the DPF environment have their own separate services file, but the entries in those files that are applicable to DB2 interpartition communication must be the same.

2.6.2 The Node Configuration File

The node configuration file (db2nodes.cfg) contains information about the database partitions and the servers on which they reside that belong to an instance. Figure 2.11 shows an example of the db2nodes.cfg file for a cluster of four UNIX servers with two partitions on each server.

In Figure 2.11, the partition number, the first column in the db2nodes.cfg file, indicates the number that identifies the database partition within DB2. You can see that there are eight partitions in total. The numbering of the partitions must be in ascending order, can start from any number, and gaps between the numbers are allowed. The numbering used is important as it will be taken into consideration in commands or SQL statements.



```
0   myservera   0
1   myservera   1
2   myserverb   0
3   myserverb   1
4   myserverc   0
5   myserverc   1
6   myserverd   0
7   myserverd   1
```

Figure 2.11 An example of the db2nodes.cfg file

The second column is the hostname or TCP/IP address of the server where the partition is created.

The third column, the logical port, is required when you create more than one partition on the same server. This column specifies the logical port for the partition within the server and must be unique within a server. In Figure 2.11, you can see the mapping between the db2nodes.cfg entries for partitions 2 and 3 for server *myserverb* and the physical machine implementation. The logical ports must also be in the same order as in the db2nodes.cfg file.

The fourth column in the db2nodes.cfg file, the netname, is required if you are using a high-speed interconnect for interpartition communication or if the resourcesetname column is used.

The fifth column in the db2nodes.cfg file, the resourcesetname, is optional. It specifies the operating system resource that the partition should be started in.

On Windows, the db2nodes.cfg file uses the *computer name* column instead of the *resourcesetname* column. The *computer name* column stores the computer name for the machine on which a partition resides. Also, the order of the columns is slightly different: partition number, hostname, computer name, logical port, netname, and resourcesetname.

The db2nodes.cfg file must be located

- Under the SQLLIB directory for the instance owner on Linux and UNIX
- Under the SQLLIB\instance_name directory on Windows

In Figure 2.10 this file would be on the Linux3 machine, as this machine is the Network File System (NFS) source server, the server whose disk(s) can be shared.

On Linux and UNIX you can edit the db2nodes.cfg file with any ASCII editor or use DB2 commands to update the file. On Windows, you can only use the **`db2ncrt`** and **`db2ndrop`** commands to create and drop database partitions; the db2nodes.cfg file should not be edited directly.

For any platform, you can also use the **`db2start`** command to add and or remove a database partition from the DB2 instance and update the db2nodes.cfg file using the **`add dbpartitionnum`** and the **`drop dbpartitionnum`** clauses respectively.

2.6.3 An Instance in the DPF Environment

Partitioning is a concept that applies to the database, not the instance; you partition a database, not an instance. In a DPF environment an instance is created once on an NFS source server machine. The instance owner's home directory is then exported to all servers where DB2 is to be run. Each partition in the database has the same characteristics: the same instance owner, password, and shared instance home directory.

On Linux and UNIX, an instance maps to an operating system user; therefore, when an instance is created, it will have its own home directory. In most installations /home/*user_name* is the home directory. All instances created on each of the participating machines in a DPF environment must use the same name and password. In addition, you must specify the home directory of the corresponding operating system user to be the same directory for all instances, which must be created on a shared file system. Figure 2.12 illustrates an example of this.

In Figure 2.12, the instance *myinst* has been created on the shared file system, and *myinst* maps to an operating system user of the same name, which in the figure has a home directory of /home/myinst. This user must be created separately in each of the participating servers, but they must share the instance home directory. As shown in Figure 2.12, all three Linux servers share /home/myinst, and it resides on a shared file system local to Linux3. Since the instance owner directory is locally stored on the Linux3 machine, this machine is considered to be the DB2 **instance-owning server**.

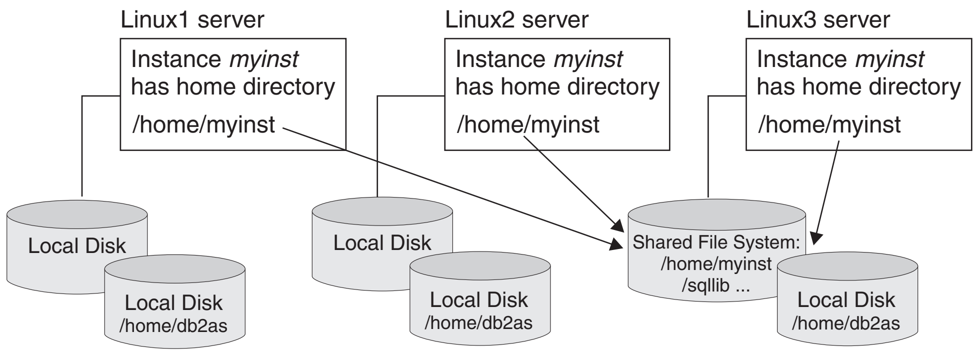

Linux1 server
Instance *myinst* has home directory /home/myinst
Linux2 server
Instance *myinst* has home directory /home/myinst
Linux3 server
Instance *myinst* has home directory /home/myinst
Local Disk
Local Disk /home/db2as
Local Disk
Local Disk /home/db2as
Shared File System: /home/myinst /sqllib ...
Local Disk /home/db2as

Figure 2.12 An instance in a partitioned environment

Figure 2.12 also shows that the Database Administration Server user db2as is created locally on each participating server in a DPF environment. There can only be one DAS per physical server regardless of the number of partitions that machine contains. The DAS user's home directory cannot be mounted on a shared file system. Alternatively, different userids and passwords can be used to create the DAS on different machines.

NOTE Make sure the passwords for the instances are the same on each of the participating machines in a DPF environment, otherwise the partitioned system will look like it is hanging because the partitions are not able to communicate.

2.6.4 Partitioning a Database

When you want to partition a database in a DPF environment, simply issue the **`CREATE DATABASE`** command as usual. For example, if the instance owner home directory is /home/myinst, when you execute this command:

```
CREATE DATABASE mydb2
```

the structure created is as shown in Figure 2.13.

```
/home
  /myinst
    /NODE0000
      /SQL00001
    /NODE0001
      /SQL00001
    /NODE0002
      /SQL00001
```

Figure 2.13 A partitioned database in a single file system

If you don't specify a path in your **CREATE DATABASE** command, by default the database is created in the directory specified by the database manager configuration parameter DFTDBPATH, which defaults to the instance owner's home directory. This partitioning is not optimal because all of the database data would reside in one file system that is shared by the other machines across a network.

We recommend that you create a directory with the same name, locally in each of the participating machines. For the environment in Figure 2.12, let's assume the directory /data has been created locally on each machine. When you execute the command:

```
CREATE DATABASE mydb2 on /data
```

the following directory structure is automatically built for you:

/data/*instance_name*/NODE*xxxx*/SQL*yyyyy*

The /data directory is specified in the **CREATE DATABASE** command, but the directory must exist *before* executing the command. *instance_name* is the name of the instance; for example, *myinst*. NODE*xxxx* distinguishes which partition you are working with, where *xxxx* represents the number of the partition specified in the db2nodes.cfg file. SQL*yyyyy* identifies the database, where *yyyyy* represents a number. If you have only one database on your system, then *yyyyy* is equal to 00001; if you have three databases on your system, you will have different directories as follows: SQL00001, SQL00002, SQL00003. To map the database names to these directories, you can review the local database directory using the command:

```
list db directory on /data
```

Inside the SQL*yyyyy* directories are subdirectories for table spaces, and within them, files containing database data—assuming all table spaces are defined as system-managed space (SMS).

Figure 2.14 illustrates a partitioned database created in the /data directory

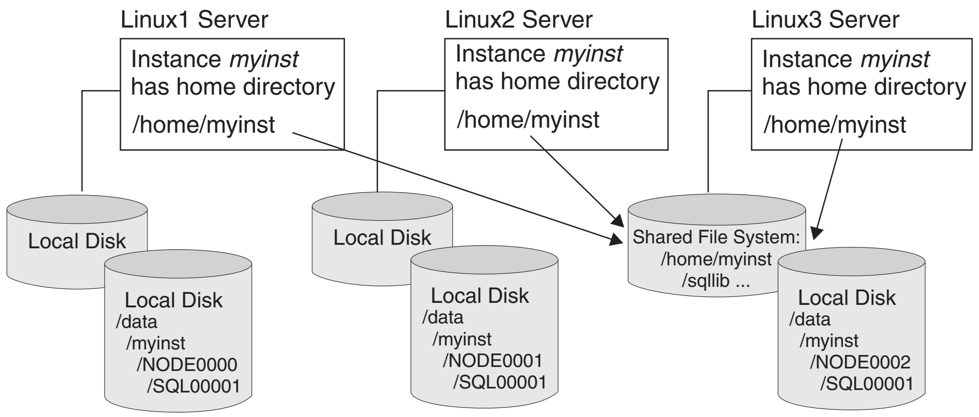

Linux1 Server
Instance *myinst* has home directory /home/myinst
Linux2 Server
Instance *myinst* has home directory /home/myinst
Linux3 Server
Instance *myinst* has home directory /home/myinst
Local Disk
Local Disk /data /myinst /NODE0000 /SQL00001
Local Disk
Local Disk /data /myinst /NODE0001 /SQL00001
Shared File System: /home/myinst /sqllib ...
Local Disk /data /myinst /NODE0002 /SQL00001

Figure 2.14 A partitioned database across several file systems

NOTE Before creating a database, be sure to change the value of the dbm cfg parameter DFTDBPATH to an existing path created locally with the same name on each of the participating machines of your DPF system. Alternatively, make sure to include this path in your **`CREATE DATABASE`** command. Similarly, to create the SAMPLE database, specify this path in the command:

```
db2sampl path
```

Partitioning a database is described in more detail in Chapter 8, The DB2 Storage Model.

2.6.5 Configuration Files in a DPF Environment

As shown in Figure 2.10, the Database Manager Configuration file (dbm cfg), system database directory, node directory, and DCS directory are all part of the instance-owning machine and are not partitioned. What about the other configuration files?

- Environment variables: Each participating server in a partitioned environment can have different environment variables.
- Global-level profile registry variable: This is stored in a file called default.env that is located in a subdirectory under the /var directory. There is a local copy of this file on each server.
- Database configuration file: This is stored in the file SQLDBCON that is located in the SQL*yyyyy* directory for the database. In a partitioned database environment, a separate SQLDBCON file is created for each partition in every database.
- The local database directory: This is stored in the file SQLDBDIR in the corresponding directory for the database. It has the same name as the system database directory, which is located under the instance directory. A separate SQLDBDIR file exists for each partition in each database.

CAUTION We strongly suggest you do *not* manually edit any of the DB2 configuration files. You should modify the files using the commands described in earlier sections.

NOTE The values of the global-level profile registry variables, database configuration file parameters, and local database directory entries should be the same for each database partition.

2.6.6 Logs in a DPF Environment

The logs on each database partition should be kept in a separate place. The database configuration parameter *Path to log files* (LOGPATH) on each partition should point to a local file system, not a shared file system. The default log path in each partition includes a NODE000*x* subdirectory. For example, the value of this parameter in the DPF system shown in Figure 2.10 could be:

- For Partition 0: /datalogs/db2inst1/NODE0000/SQL00001/SQLOGDIR/
- For Partition 1: /datalogs/db2inst1/NODE0001/SQL00001/SQLOGDIR/
- For Partition 2: /datalogs/db2inst1/NODE0002/SQL00001/SQLOGDIR/

To change the path for the logs, update the database configuration parameter NEWLOGPATH.

2.6.7 The Catalog Partition

As stated previously, when you create a database, several table spaces are created by default. One of them, the catalog table space SYSCATSPACE, contains the DB2 system catalogs. In a partitioned environment SYSCATSPACE is not partitioned, but resides on one partition known as the **catalog partition**. The partition from which the `CREATE DATABASE` command is issued becomes the catalog partition for the new database. All access to system tables must go through this database partition. Figure 2.10 shows SYSCATSPACE residing on server Linux1, so the `CREATE DATABASE` command was issued from this server.

For an existing database, you can determine which partition is the catalog partition by issuing the command `list db directory`. The output of this command has the field *Catalog database partition number* for each of the entries, which indicates the catalog partition number for that database.

2.6.8 Partition Groups

A **partition group** is a logical layer that provides for the grouping of one or more database partitions. A database partition can belong to more than one partition group. When a database is created, DB2 creates three default partition groups, and these partition groups cannot be dropped.

- **IBMDEFAULTGROUP**: This is the default partition group for any table you create. It contains all database partitions defined in the db2nodes.cfg file. This partition group cannot be modified. Table space USERSPACE1 is created in this partition group.
- **IBMTEMPGROUP**: This partition group is used by all system temporary tables. It contains all database partitions defined in the db2nodes.cfg file. Table space TEMPSPACE1 is created in this partition.
- **IBMCATGROUP**: This partition group contains the catalog tables (table space SYSCATSPACE). It only includes the database's catalog partition. This partition group cannot be modified.

To create new database partition groups, use the `CREATE DATABASE PARTITION GROUP` statement. This statement creates the database partition group within the database, assigns database partitions that you specified to the partition group, and records the partition group definition in the database system catalog tables.

The following statement creates partition group *pgrpall* on all partitions specified in the db2nodes.cfg file:

```
CREATE DATABASE PARTITION GROUP pgrpall ON ALL DBPARTITIONNUMS
```

To create a database partition group *pg23* consisting of partitions 2 and 3, issue this command:

```
CREATE DATABASE PARTITION GROUP pg23 ON DBPARTITIONNUMS (2,3)
```

Other relevant partition group statements/commands are:

- **`ALTER DATABASE PARTITION GROUP`** (statement to add or drop a partition in the group)
- **`DROP DATABASE PARTITION GROUP`** (statement to drop a partition group)
- **`LIST DATABASE PARTITION GROUPS`** (command to list all your partition groups; note that IBMTEMPGROUP is never listed)

2.6.9 Buffer Pools in a DPF Environment

Figure 2.10 shows buffer pools defined across all of the database partitions. Interpreting this figure for buffer pools is different than for the other objects, because the data cached in the buffer pools is not partitioned as the figure implies. Each buffer pool in a DPF environment holds data only from the database partition where the buffer pool is located.

You can create a buffer pool in a partition group using the **`CREATE BUFFERPOOL`** statement with the **`DATABASE PARTITION GROUP`** clause. This means that you have the flexibility to define the buffer pool on the specific partitions defined in the partition group. In addition, the size of the buffer pool on each partition in the partition group can be different. The following statement will create buffer pool *bpool_1* in partition group *pg234*, which consists of partitions 2, 3, and 4.

```
CREATE BUFFERPOOL bpool_1 DATABASE PARTITION GROUP pg234
       SIZE 10000
       EXCEPT ON DBPARTITIONNUM (3 TO 4) SIZE 5000
```

Partition 2 in partition group *pg234* will have a buffer pool *bpool_1* defined with a size of 10,000 pages, and Partitions 3 and 4 will have a buffer pool of size 5,000 pages.

As an analogy, think of it as if you were issuing the **`CREATE BUFFERPOOL`** statement on each partition separately, with the same buffer pool name for each partition but with different sizes. That is:

- On partition 2: **`CREATE BUFFERPOOL bpool_1 SIZE 10000`**
- On partition 3: **`CREATE BUFFERPOOL bpool_1 SIZE 5000`**
- On partition 4: **`CREATE BUFFERPOOL bpool_1 SIZE 5000`**

Note that we use these statements only to clarify the analogy; they will not work as written. Executing each of these commands as shown will attempt to create the same buffer pool on all partitions. It is not equivalent to using the **`DATABASE PARTITION GROUP`** clause of the **`CREATE BUFFERPOOL`** statement.

Buffer pools can also be associated to several partition groups. This means that the buffer pool definition will be applied to the partitions in those partition groups.

2.6.10 Table Spaces in a Partitioned Database Environment

You can create a table space in specific partitions, associating it to a partition group, by using the **CREATE TABLESPACE** statement with the **IN DATABASE PARTITION GROUP** clause. This allows users to have flexibility as to which partitions will actually be storing their tables. In a partitioned database environment with three servers, one partition per server, the statement:

```
CREATE TABLESPACE mytbls IN DATABASE PARTITION GROUP pg234
       MANAGED BY SYSTEM USING ('/data')
       BUFFERPOOL bpool_1
```

creates the table space *mytbls*, which spans partitions 2, 3, and 4 (assuming *pg234* is a partition group consisting of these partitions). In addition, the table space is associated with buffer pool *bpool_1* defined earlier. Note that creating a table space would fail if you provide conflicting partition information between the table space and the associated buffer pool. For example, if *bpool_1* was created for partitions 5 and 6, and table space *mytbls* was created for partitions 2, 3, and 4, you would get an error message when trying to create this table space.

2.6.11 The Coordinator Partition

In general, each database connection has a corresponding DB2 agent handling the application connection. An **agent** can be thought of as a process (Linux/UNIX) or thread (Windows) that performs DB2 work on behalf of the application. There are different types of agents. One of them, the coordinator agent, communicates with the application, receiving requests and sending replies. It can either satisfy the request itself or delegate the work to multiple subagents to work on the request.

The **coordinator partition** of a given application is the partition where the coordinator agent exists. You use the **SET CLIENT CONNECT_NODE** command to set the partition that is to be the coordinator partition. Any partition can potentially be a coordinator, so in Figure 2.10 we do not label any particular partition as the coordinator node. If you would like to know more about DB2 agents and the DB2 process model, refer to Chapter 14, The DB2 Process Model.

2.6.12 Issuing Commands and SQL Statements in a DPF Environment

Imagine that you have twenty physical servers, with two database partitions on each. Issuing individual commands to each physical server or partition would be quite a task. Fortunately, DB2 provides a command that executes on all database partitions.

2.6.12.1 The db2_all command

Use the **db2_all** command when you want to execute a command or SQL statement against all database partitions. For example, to change the db cfg parameter LOGFILSIZ for the database *sample* in all partitions, you would use:

```
db2_all ";db2 UPDATE DB CFG FOR sample USING LOGFILSIZ 500"
```

When the semicolon (;) character is placed before the command or statement, the request runs in parallel on all partitions.

NOTE In partitioned environments, the operating system command **rah** performs commands on all servers simultaneously. The **rah** command works per server, while the **db2_all** command works per database partition. The **rah** and **db2_all** commands use the same characters. For more information about the **rah** command, refer to your operating system manuals.

2.6.12.2 Using Database Partition Expressions

In a partitioned database, database partition expressions can be used to generate values based on the partition number found in the db2nodes.cfg file. This is particularly useful when you have a large number of database partitions and when more than one database partition resides on the same physical machine, because the same device or path cannot be specified for all partitions. You can manually specify a unique container for each database partition or use database partition expressions. The following example illustrates the use of database partition expressions.

On Linux/UNIX, here are sample contents of a db2nodes.cfg file:

```
0     myservera     0
1     myservera     1
2     myserverb     0
3     myserverb     1
```

This shows two servers with two database partitions each. The command:

```
CREATE TABLESPACE ts2
  MANAGED BY DATABASE USING
  (file '/data/TS2/container $N+100' 5000)
```

creates the following containers:

- /data/TS2/container100 on database partition 0
- /data/TS2/container101 on database partition 1
- /data/TS2/container102 on database partition 2
- /data/TS2/container103 on database partition 3

You specify a database partition expression with the argument **$N** (note that there must be a space before **$N in** the command). Table 2.1 shows other arguments for creating containers. Operators are evaluated from left to right, and **%** represents the modulus (the remainder of a division). Assuming the partition number to be evaluated is 3, the value column in Table 2.1 shows the result of resolving the database partition expression.

Table 2.1 Database Partition Expressions

Database Partition Expressions	Example	Value
[blank]$N	$N	3
[blank]$N+[number]	$N+500	503
[blank]$N%[number]	$N%2	1
[blank]$N+[number]%[number]	$N+15%13	5
[blank]$N%[number]+[number]	$N%2+20	21

2.6.13 The DB2NODE Environment Variable

In section 2.3, The DB2 Environment, we talked about the DB2INSTANCE environment variable used to switch between instances in your database system. The DB2NODE environment variable is used in a similar way, but to switch between partitions on your DPF system. By default, the active partition is the one defined with the logical port number of zero (0) in the db2nodes.cfg file for a server. To switch the active partition, change the value of the DB2NODE variable using the **`SET`** command on Windows and the **`export`** command on Linux/UNIX. Be sure to issue a **`terminate`** command for all connections from any partition to your database after changing this variable or the change will not take effect.

Using the settings for the db2nodes.cfg file shown in Table 2.2, you have four servers, each with two logical partitions. If you log on to server *myserverb*, any commands you execute will affect partition 2, which is the one with logical port of zero on that server, and the default coordinator partition for that server.

Table 2.2 Sample Partition Information

Partition	Server Name	Logical Port
0	myservera	0
1	myservera	1
2	myserverb	0
3	myserverb	1
4	myserverc	0
5	myserverc	1
6	myserverd	0
7	myserverd	1

If you would like to make partition 0 the active partition, make this change on a Linux/UNIX system:

```
DB2NODE=0
export DB2NODE
db2 terminate
```

NOTE You must issue the **`terminate`** command, even if there aren't any connections to any partitions.

Note that partition 0 is on server *myservera*. Even if you are connected to *myserverb*, you can make a partition on *myservera* the active one. To determine which is your active partition, you can issue this statement after connecting to a database:

```
db2 "values (current dbpartitionnum)"
```

2.6.14 Partitioning Maps and Partitioning Keys

By now you should have a good grasp of how to set up a DPF environment. It is now time to understand how DB2 distributes data across the partitions. Figure 2.15 shows an example of this distribution.

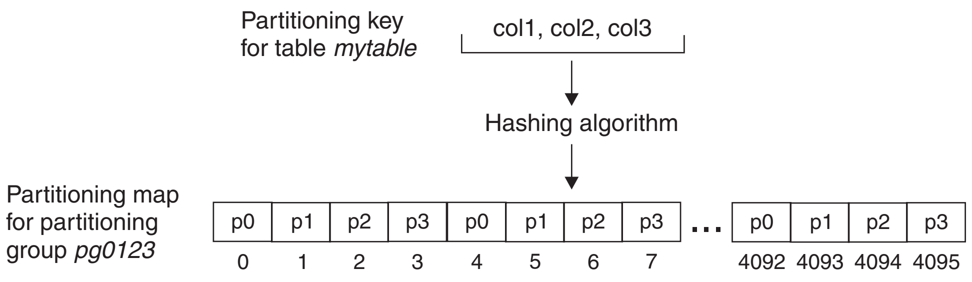


Figure 2.15 Distributing data rows in a DPF environment

A **partitioning map** is an internally generated array containing 4096 entries for multipartition database partition groups or a single entry for single-partition database partition groups. The partition numbers of the database partition group are specified in a round-robin fashion in the array.

A **partitioning key** is a column (or group of columns) that determines the partition on which a particular row of data is physically stored. You define a partitioning key explicitly using the **`CREATE TABLE`** statement with the PARTITIONING KEY clause.

When you create or modify a database partition group, a partitioning map is associated with it. A partitioning map in conjunction with a partitioning key and a hashing algorithm determine which database partition will store a given row of data.

For the example in Figure 2.15, let's assume partition group *pg0123* has been defined on partitions 0, 1, 2, and 3. An associated partitioning map is automatically created. This map is an array with 4096 entries containing the values 0, 1, 2, 3, 0, 1, 2, 3. . . . (note that this is shown in Figure 2.15 as p0, p1, p2, p3, p0, p1, p2, p3 . . . to distinguish them from the array entry numbers). Let's also assume table *mytable* has been created with a partitioning key consisting of columns col1, col2, and col3. For each row, the partitioning key column values are passed to the hashing algorithm, which returns an output number from 0 to 4095. This number corresponds to one of the entries in the array that contains the value of the partition number where the row is to be stored. In Figure 2.15, if the hashing algorithm had returned an output value of 7, the row would have been stored in partition *p3*.

2.7 Case Study: DB2 with DPF Environment

Now that you are familiar with DPF, let's review some of the concepts discussed using a simple case study.

Your company is expanding, and it recently acquired two other firms. Since the amount of data will be increased by approximately threefold, you are wondering if your current single-partition DB2 database server will be able to handle the load, or if DB2 with DPF will be required. You are not too familiar with DB2 with DPF, so you decide to play around with it using your test machines: two SMP machines running Linux with four processors each. The previous DBA, who has left the company, had installed DB2 UDB ESE with DPF on these machines. Fortunately, he left a diagram with his design, shown in Figure 2.16.

Figure 2.16 is a combined physical and logical design. When you validate the correctness of the diagram with your system, you note that database *mydb1* has been dropped, so you decide to rebuild this database as practice. The instance *db2inst1* is still there, as are other databases. These are the steps you follow.

1. Open two telnet sessions, one for each server. From one of the sessions you issue the commands **`db2stop`** followed by **`db2start`**, as shown in Figure 2.17.

   The first thing you note is that there is no need to issue these two commands on each partition; issuing them on any partition once will affect all partitions. You also can tell that there are four partitions, since you received a message from each of them.
2. Review the db2nodes.cfg file to understand the configuration of your partitions (see Figure 2.18). Using operating system commands, you determine that the home directory for instance *db2inst1* is /home/db2inst1. The db2nodes.cfg file is stored in the directory /home/db2inst1/sqllib.

   Figure 2.18 shows there are four partitions, two per server. The server host names are *aries* and *saturn*.
3. Create the database *mydb1*. Since you want partition 0 to be your catalog partition, you must issue the **`CREATE DATABASE`** command from partition 0. You issue the statement **`db2 "values (current dbpartitionnum)"`** to determine which partition is currently active and find out that partition 3 is the active partition (see Figure 2.19).

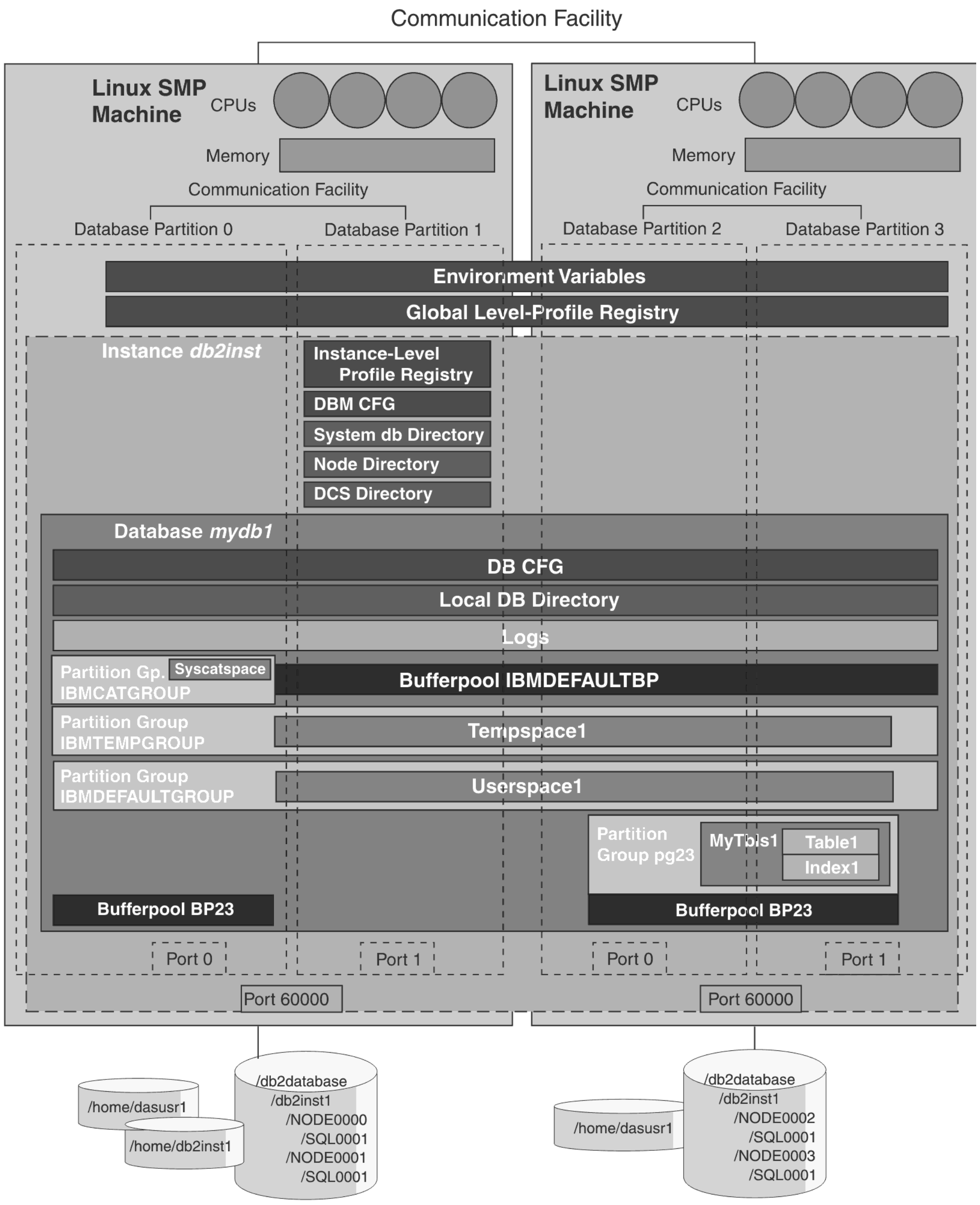


Figure 2.16 DB2 UDB ESE with DPF design

```
[db2inst1@aries db2inst1]$ db2stop
05-18-2004 23:44:42     3   0   SQL1064N  DB2STOP processing was successful.
05-18-2004 23:44:43     1   0   SQL1064N  DB2STOP processing was successful.
05-18-2004 23:44:44     2   0   SQL1064N  DB2STOP processing was successful.
05-18-2004 23:44:44     0   0   SQL1064N  DB2STOP processing was successful.
SQL1064N  DB2STOP processing was successful.

[db2inst1@aries db2inst1]$ db2start
05-18-2004 23:44:51     1   0   SQL1063N  DB2START processing was successful.
05-18-2004 23:44:51     0   0   SQL1063N  DB2START processing was successful.
05-18-2004 23:44:52     3   0   SQL1063N  DB2START processing was successful.
05-18-2004 23:44:53     2   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.

[db2inst1@aries db2inst1]$
```

Figure 2.17 Running the db2stop and db2start commands

```
[db2inst1@aries sqllib]$ pwd
/home/db2inst1/sqllib

[db2inst1@aries sqllib]$ more db2nodes.cfg
0 aries.myacme.com 0
1 aries.myacme.com 1
2 saturn.myacme.com 0
3 saturn.myacme.com 1

[db2inst1@aries sqllib]$
```

Figure 2.18 A sample db2nodes.cfg file

```
[db2inst1@saturn db2inst1]$ db2 "values (current dbpartitionnum)"

1
-----------
          3

  1 record(s) selected.
```

Figure 2.19 Determining the active partition

4. Next, you change the DB2NODE environment variable to zero (0) as follows (see Figure 2.20):

```
DB2NODE=0
export DB2NODE
db2 terminate
```

```
[db2inst1@saturn db2inst1]$ DB2NODE=0
[db2inst1@saturn db2inst1]$ export DB2NODE
[db2inst1@saturn db2inst1]$ db2 terminate
DB20000I  The TERMINATE command completed successfully.

[db2inst1@saturn db2inst1]$ db2 list applications
SQL1611W  No data was returned by Database System Monitor. SQLSTATE=00000

[db2inst1@saturn db2inst1]$ db2 create db mydb1 on /db2database
DB20000I  The CREATE DATABASE command completed successfully.

[db2inst1@saturn db2inst1]$ db2 connect to mydb1

   Database Connection Information

 Database server        = DB2/LINUX 8.1.2
 SQL authorization ID   = DB2INST1
 Local database alias   = MYDB1

[db2inst1@saturn db2inst1]$ db2 "values (current dbpartitionnum)"

1
-----------
          0

  1 record(s) selected.

[db2inst1@saturn db2inst1]$
```

Figure 2.20 Switching the active partition, then creating a database

In the **`CREATE DATABASE`** command you specify the path, /db2database in this example, which is an existing path that has been created locally on all servers so that the data is spread across them.

5. To confirm that partition 0 is indeed the catalog partition, simply issue a **`list db directory`** command and look for the *Catalog database partition number* field under the entry for the *mydb1* database. Alternatively, issue a **`list tablespaces`** command from each partition. The SYSCATSPACE table space will be listed only on the catalog partition.
6. Create partition group *pg23* on partitions 2 and 3. Figure 2.21 shows how to accomplish this and how to list your partition groups. Remember that this does not list IBMTEMPGROUP.
7. Create and manage your buffer pools. Issue this statement to create buffer pool *BP23* on partition group *pg23*:

```
db2 "create bufferpool BP23 database partition group pg23 size 500"
```

```
[db2inst1@saturn db2inst1]$ db2 "create database partition group pg23 on
dbpartitionnum (2 to 3)"
DB20000I  The SQL command completed successfully.

[db2inst1@saturn db2inst1]$ db2 "list database partition groups"

DATABASE PARTITION GROUP
----------------------------
IBMCATGROUP
IBMDEFAULTGROUP
PG23

  3 record(s) selected.

[db2inst1@saturn db2inst1]$
```

Figure 2.21 Creating partition group pg23

Figure 2.22 shows this statement. It also shows you how to associate this buffer pool to another partition group using the **`ALTER BUFFERPOOL`** statement.

To list your buffer pools and associated partition groups, you can query the SYSCAT.BUFFERPOOLS catalog view, also shown in Figure 2.22.

```
[db2inst1@saturn db2inst1]$ db2 "create bufferpool BP23 database partition group
pg23 size 500"
DB20000I  The SQL command completed successfully.

[db2inst1@saturn db2inst1]$ db2 "alter bufferpool BP23 add database partition group
IBMCATGROUP"
DB20000I  The SQL command completed successfully.

[db2inst1@saturn db2inst1]$ db2 "select bpname, ngname from syscat.bufferpools"

BPNAME                                          NGNAME
--------------------------------------------------------------------------------
IBMDEFAULTBP                                    -

BP23                                            PG23

BP23                                            IBMCATGROUP

  3 record(s) selected.

[db2inst1@saturn db2inst1]$
```

Figure 2.22 Managing buffer pools

Note that a buffer pool can be associated with any partition group. Its definition will be applied to all the partitions in the partition group, and you can specify different sizes on the partitions if required.

8. Create the table space *mytbls1*:

```
db2 "create tablespace mytbls1 in database partition group pg23
         managed by system using ('/data') bufferpool bp23"
```

9. Create table *table1* in table space *mytbls1* with a partitioning key of col1 and col2:

```
db2 "create table table1 (col1 int, col2 int, col3 char(10))
            in mytbls1
            partitioning key (col1, col2)"
```

10. Create the index *index1*. Note that this doesn't have any syntax specific to a DPF environment:

```
db2 "create index index1 on table1 (col1, col2)"
```

    The index will be constructed on each partition for its subset of rows.

11. Test the **`db2_all`** command to update the database configuration file for all partitions with one command. Figure 2.23 shows an example of this.

```
[db2inst1@aries sqllib]$ db2 get db cfg for mydb1 | grep LOGFILSIZ
 Log file size (4KB)                         (LOGFILSIZ) = 1000
[db2inst1@aries sqllib]$ db2_all "db2 update db cfg for mydb1 using LOGFILSIZ 500"

DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
aries.myacme.com: db2 update db cfg for mydb1 using LOGFILSIZ 500 completed ok

DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
aries.myacme.com: db2 update db cfg for mydb1 using LOGFILSIZ 500 completed ok

DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
saturn.myacme.com: db2 update db cfg for mydb1 using LOGFILSIZ 500 completed ok

DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
saturn.myacme.com: db2 update db cfg for mydb1 using LOGFILSIZ 500 completed ok
[db2inst1@aries sqllib]$ db2 get db cfg for mydb1 | grep LOGFILSIZ
 Log file size (4KB)                         (LOGFILSIZ) = 500
[db2inst1@aries sqllib]$
```

Figure 2.23 Using db2_all to update the db cfg file

And that's it! In this case study you have reviewed some basic statements and commands applicable to the DPF environment. You reviewed the **`db2stop`** and **`db2start`** commands, determined and switched the active partition, and created a database, a partition group, a buffer pool, a table space, a table with a partitioning key, and an index. You also used the **`db2_all`** command to update a database configuration file parameter.

2.8 Summary

This chapter provided an overview of the DB2 core concepts using a "big picture" approach. It introduced SQL statements and their classification in Data Definition Language (DDL), Data Manipulation Language (DML), and Data Control Language (DCL).

DB2 commands are classified into two groups—system commands and CLP commands—and several examples were provided, like the command to start an instance, `db2start`.

You need an interface to issue SQL statements and commands to the DB2 engine. This interface was provided by using the DB2 tools available with the product. Two text-based interfaces were mentioned, the Command Line Processor (CLP) and the Command Window. The Control Center was noted as being the most important administration graphical tool.

The chapter introduced the concepts of instances, databases, table spaces, buffer pools, logs, tables, indexes, and other database objects in a single partition system. There are different levels of configuration for the DB2 environment: the environment variables, the DB2 registry variables, and the configuration parameters at the instance (dbm cfg) and database (db cfg) levels. DB2 has federation support for queries using tables from other databases in the DB2 family. The chapter also covered database partition, catalog partition, coordinator node, and partitioning map on a multipartition system.

Two case studies reviewed the single-partition and multipartition environments respectively, which should help you understand the topics discussed in the chapter.

2.9 Review Questions

1. How are DB2 commands classified?
2. What is a quick way to obtain help information for a command?
3. What is the difference between the Information Center tool and simply reviewing the DB2 manuals?
4. What command is used to create a DB2 instance?
5. How many table spaces are automatically created by the `CREATE DATABASE` command?
6. What command can be used to get a list of all instances on your server?
7. What is the default instance that is created on Windows?
8. Is the DAS required to be running to set up a remote connection between a DB2 client and a DB2 server?
9. How can the DB2 environment be configured?
10. How is the local database directory populated?
11. Which of the following commands will start your DB2 instance?
    A. startdb
    B. db2 start
    C. db2start
    D. start db2
12. Which of the following commands will list all of the registry variables that are set on your server?
    A. db2set –a
    B. db2set –all

C. db2set –lr
D. db2set -ltr

13. Say you are running DB2 on a Windows server with only one hard drive (C:). If the DB2 instance is dropped using the `db2idrop` command, after recreating the DB2 instance, which of the following commands will list the databases you had prior to dropping the instance?
A. list databases
B. list db directory
C. list db directory all
D. list db directory on C:

14. If the `list db directory on C:` command returns the following:

```
Database alias                         = SAMPLE
Database name                          = SAMPLE
Database directory                     = SQL00001
Database release level                 = a.00
Comment                                =
Directory entry type                   = Home
Catalog database partition number      = 0
Database partition number              = 0
```

which of the following commands must be run before you can access tables in the database?
A. catalog db sample
B. catalog db sample on local
C. catalog db sample on SQL00001
D. catalog db sample on C:

15. If there are two DB2 instances on your Linux server, *inst1* and *inst2*, and if your default DB2 instance is *inst1*, which of the following commands allows you to connect to databases in the *inst2* instance?
A. export inst2
B. export instance=inst2
C. export db2instance=inst2
D. connect to inst2

16. Which of the following DB2 registry variables optimizes interpartition communication if you have multiple partitions on a single server?
A. DB2_OPTIMIZE_COMM
B. DB2_FORCE_COMM
C. DB2_USE_FCM_BP
D. DB2_FORCE_FCM_BP

17. Which of the following tools is used to run commands on all partitions in a multipartition DB2 database?
A. db2_part
B. db2_all
C. db2_allpart
D. db2

18. Which of the following allows federated support in your server?
- **A.** db2 update db cfg for federation using FEDERATED ON
- **B.** db2 update dbm cfg using FEDERATED YES
- **C.** db2 update dbm cfg using NICKNAME YES
- **D.** db2 update dbm cfg using NICKNAME, WRAPPER, SERVER, USER MAPPING YES

19. Which environment variable needs to be updated to change the active logical database partition?
- **A.** DB2INSTANCE
- **B.** DB2PARTITION
- **C.** DB2NODE
- **D.** DB2PARTITIONNUMBER

20. Which of the following statements can be used to determine the value of the current active database partition?
- **A.** values (current dbpartitionnum)
- **B.** values (current db2node)
- **C.** values (current db2partition)
- **D.** values (current partitionnum)