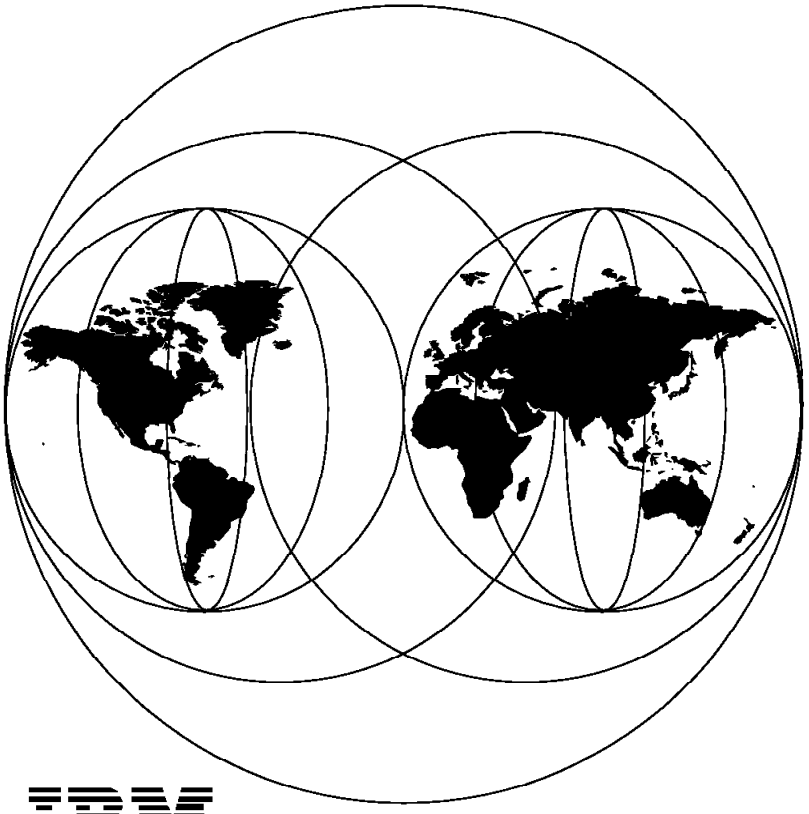
International Technical Support Organization

SG24-4691-00

DATABASE 2 for AIX
Programming Interfaces

May 1996



IBM

International Technical Support Organization
Austin Center

International Technical Support Organization

SG24-4691-00

DATABASE 2 for AIX
Programming Interfaces

May 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information under “Special Notices” on page xiii.

First Edition (May 1996)

This edition applies to DATABASE 2 Version 2.1.1 for use with the AIX operating system.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader's feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. JN9B Building 045 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

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

Abstract

This document is unique in its detailed coverage of the different application programming interfaces that are available for use with DATABASE 2 for AIX. It focuses on introducing the interfaces and describing how to use each of them. Sample applications have been provided and discussed with each interface. These sample applications can be used as a starting point or template for new applications. Information is provided about each of the interfaces, and references are made to documentation where further details may be found.

This document was written for the application programmer who is about to develop a new application that will access DB2 databases. Some knowledge of application programming and DB2 for AIX is assumed.

(170 pages)

Contents

Figures

Tables

Special Notices

This publication is intended to help the application programmer and the system administrator decide on the best interface to use when developing new database applications. The information in this publication is not intended as the specification of any programming interfaces that are provided by the DATABASE 2 Common Server products. It is a guideline to the use and implementation of these interfaces. See the PUBLICATIONS section of the IBM Programming Announcement for DB2 for AIX for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

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

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX
AT
BookManager
BookMaster
DATABASE 2
DataJoiner
DB2
DB2/6000
DRDA
IBM
MVS/ESA
OfficeVision
OS/2
OS/400
SQL/DS
VisualAge
VisualGen

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

AMI	American Megatrends, Incorporated
Ami Pro	Lotus Development Corporation
C + +	AT&T, Incorporated
INTERSOLV	INTERSOLV Corporation
WordPerfect	WordPerfect Corporation
X/Open	X/Open Company Limited

Other trademarks are trademarks of their respective companies.

Preface

This document is intended to assist an application delveloper who is about to start development of an application that will access information contained within DATABASE 2 for AIX databases. It contains descriptions and sample applications for the different methods that may be used to access database tables. There is also coverage of the database extender products and discussion on developing applications that will make use of the extender search capabilities.

This document is intended as a guideline to help you decide which interface into DB2 for AIX will best suit your needs. Once the interface has been decided, the book will prove useful as a starting point for your applications.

How This Redbook is Organized

The redbook is organized as follows:

- Chapter 1, “Programming Interfaces”

  This chapter provides an overview to the different programming interfaces and what factors need to be considered.

- Chapter 2, “Command Line Processor”

  The Command Line Processor allows you to enter database commands and SQL statements from the command prompt. This chapter discusses the Command Line Processor and how you may use it from within different shell scripts.

- Chapter 3, “World Wide Web”

  The World Wide Web is growing, and it is opening a new way of accessing databases around the world. DB2 may be accessed through many of the Web browsers. This chapter discusses the different machanisms used in accessing DB2 from the Internet and World Wide Web, along with the issues that must be considered when providing this access.

- Chapter 4, “Call Level Interface”

  The call level interface may be used from applications to access databases. This chapter discusses the Call Level Interface and how it is used.

- Chapter 5, “Open Database Connectivity (ODBC)”

  This chapter covers the configuration of the Open Database Connectivity (ODBC) environment. There are a number of different ways to access DB2 databases using the ODBC interface. This chapter looks at each of these methods and the configuration required for each method.

- Chapter 6, “Database Extenders”

  The database extender products may be used in conjunction with DB2 to provide some additional functionality. This chapter describes the extender products and how they are used.

Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

- *Application Programming Guide - for common servers*, S20H-4643-01
- *API Reference - for common servers*, S20H-4984-00
- *Call Level Interface Guide and Reference - for common servers*, S20H-4644-01

International Technical Support Organization Publications

- *Access to the DB2 Family with ODBC*, GG24-4333-00

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

How Customers Can Get Redbooks and Other ITSO Deliverables

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **IBMLINK**

  Registered customers have access to PUBORDER to order hardcopy and to the REDBOOKS disk to obtain BookManager BOOKs.

- **IBM Bookshop** — send orders to:

  usib6fpl@ibmmail.com (United States)
  bookshop@dk.ibm.com (Outside United States)

- **Telephone orders**

  1-800-879-2755 (United States)
  354-9408 (Australia)
  359-2-731076 (Bulgaria)
  42-2-67106-250 (Czech Republic)
  593-2-5651-00 (Ecuador)
  03-69-78901 (Israel)
  905-627-1163 (Mexico)
  064-4-57659-36 (New Zealand)
  027-011-320-9299 (South Africa)
  0256-478166 (United Kingdom)
  32-2-225-3738 (Belgium)
  1-800-IBM-CALL (Canada)
  45-934545 (Denmark)
  01805-5090 (Germany)
  0462-73-6669 (Japan)
  31-20513-5100 (The Netherlands)
  507-639977 (Panama)

- **Mail Orders** — send orders to:

  IBM Publications
  P.O. Box 9046
  Boulder, CO 80301-9191
  USA

  IBM Direct Services
  Sortemosevej 21,
  3450 Allerod
  Denmark

- **Fax** — send orders to:

  1-800-445-9269 (United States)
  32-2-225-3478 (Belgium)
  905-316-7210 (Canada)
  0256-843173 (United Kingdom)
  359-2-730235 (Bulgaria)
  42-2-67106-402 (Czech Republic)

593-2-5651-45 (Ecuador)
03-69-59985 (Israel)
31-20513-3296 (The Netherlands)
507-693604 (Panama)
07032-15-3300 (Germany)
0462-73-7313 (Japan)
064-4-57659-16 (New Zealand)
027-011-320-9113 (South Africa)

- **1-800-IBM-4FAX (United States only)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services**

  Send note to softwareshop@vnet.ibm.com

- **Redbooks Home Page on the World Wide Web**

  http://www.redbooks.ibm.com/redbooks

- **E-mail (Internet)**

  Send note to redbook@vnet.ibm.com

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How IBM Employees Can Get Redbooks and ITSO Deliverables

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States

- **GOPHER link to the Internet**

  Type GOPHER
  Select IBM GOPHER SERVERS
  Select ITSO GOPHER SERVER for Redbooks

- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET GG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET GG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

http://w3.itso.ibm.com/redbooks/redbooks.html

- **ITSO4USA category on INEWS**
- **IBM Bookshop** — send orders to:

  USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

Acknowledgments

This project was designed and managed by:

Frank Rusconi
International Technical Support Organization, Austin Center

The authors of this document are:

Ada Lau Azurza
IBM Peru

Vera Puspasari S.
Multipolar Corporation, Indonesia

Frank Rusconi
IBM Austin

This publication is the result of a residency conducted at the International Technical Support Organization, Austin Center.

Thanks to the following people for the invaluable advice and guidance provided in the production of this document:

Calene Janacek
International Technical Support Organization, Austin Center

David Oberstadt
IBM San Jose

Serge Limoges
IBM Toronto

Doug Free
IBM Dallas, Education and Training

Marcus Brewer, Editor
International Technical Support Organization, Austin Center

Chapter 1. Programming Interfaces

This chapter looks at the different programming interfaces available for DB2 and discusses the advantages and disadvantages of using each. These interfaces are covered in more detail throughout the book. This chapter is designed to give an overview which should help you decide which interface is best suited for your environment.

1.1 Overview

There are many factors that can influence your decision on which interface to use in your DB2 application. These factors might include the level of security required within the application or how many users are going to use the application. As an example, you might require an application to perform some system administration task. This application may only be used once a month by a skilled database administrator. For this type of application, you would not invest a large amount of time and resources developing a sophisticated application. Instead, you might write a script that asks for the basic information required and then automates the execution of the statements or commands. The security for this application might be as simple as placing it in a secure directory where users would not be able to execute it. On the other hand, an application that has many users of varying skill levels and authorities might require a large amount of built-in security, or integrity checks, and present the user with help about the type of information the application requires in order to execute correctly. This type of application might require many hours to develop and maintain.

This chapter discusses the following interfaces and looks at the types of applications they are best suited for:

- Command Line Processor
- World Wide Web
- Embedded SQL
- Call Level Interface
- Open Database Connectivity
- Database Extenders

1.2 Command Line Processor

The Command Line Processor (CLP) is supplied with many of the DB2 for Common Server products. The CLP enables you to enter any DB2 database command or SQL statement. The CLP may be used interactively or called from an application, the most common of which are scripting languages such as Korn, Bourne or Perl scripts.

Developing an application using the Command Line Processor is quick and requires little effort by a programmer or system administrator. However, there is little room for integrity checking of data or error recovery should the application fail. This is mainly due to the limited amount of diagnostic information returned by the Command Line Processor.

The Command Line Processor is an extremely useful tool for prototyping your SQL statements or for use in scripts to perform DB2 administration tasks and simple SQL statements.

Chapter 2, “Command Line Processor” on page 5, discusses the Command Line Processor in more detail and how it may be used.

1.3 World Wide Web

First we saw a move to place computers in the homes and offices of people around the world, and now these computers are being linked together through the Internet. This growing network of computers is known as the World Wide Web (WWW), and now we are looking at ways to provide its users with up-to-date information about our business in a quick and simple manner.

DB2 allows us to provide WWW access to the databases in two ways. The first method is by using the HyperText Markup Language (HTML) as a front-end to scripts or other applications that access the databases. This method can be quite a complicated matter for anything more than simple queries.

The process of using HTML to present SQL queries, or database commands, to the DB2 server has been made much simpler through the use of the DB2 WWW Connection Program. The DB2 WWW Connection Program simplifies the process by automatically passing the required information from DB2 to HTML and from HTML to DB2 through the use of variables. These variables can be used by SQL statements to form a query or some other type of SQL statement. The variables also can be used by HTML to help format the information returned by DB2 when the statement is executed.

By allowing a database to be accessed from the WWW, you can provide information to the users on the Internet through a graphical interface, as well as to anyone who is connected over your own local network. This allows you to quickly develop graphical applications for users, using little more than basic SQL and HTML knowledge.

1.4 Embedded SQL

There are many situations where your application may need to perform more complex tasks than can be handled by a simple shell script or WWW interface. In these situations, you may need to consider writing your applications in a particular programming language, such as C or COBOL.

If you choose to use one of these languages, you also have a couple of options on how the applications are going to allow the users to access the information contained within the database. One of these options is to use Embedded SQL.

Embedded SQL, as the name implies, refers to placing SQL statements within your application. These statements are pre-compiled into code for the language you are using and then compiled into an executable application with the rest of your program. The advantages of using Embedded SQL is the added flexibility you have over performing data integrity checking and security checking at the application level. It is also possible to leave the security or data integrity checking to the database manager and then code your application to handle any error situations that may occur due to invalid data or insufficient privilege.

There are two different types of Embedded SQL, these are:

- Static SQL
- Dynamic SQL

Static SQL is an SQL statement where the statement structure is fully known when the application is being compiled. This means that the statement can be optimized and access plans defined during the application's compilation and binding. This also means that the creation of the access strategy is not a cost during execution.

Dynamic SQL statements are not completely known at compilation time and will need to be optimized and have access plans created during the execution of the application. This means that applications using dynamic SQL may have higher overhead during their execution, but they provide the added flexibility of allowing the user to determine the statement by providing information that will be used in it. Another benefit of dynamic SQL is that the most recent statistics available through the `runstats` utility are used by the optimizer.

Authorization within the database also is determined at different times, depending on your choice of dynamic or static SQL. Since static SQL statements are optimized and the access plans determined at compilation, so is the authorization. This means that if a user is able to execute the application, the user will inherit the authorization of the user who compiled and bound the application to the database.

Dynamic SQL does not get bound or optimized during compilation; so access plans and authorization were determined at run time. This requires the user executing the application to have the appropriate authorizations on the database object being accessed.

The features provided with embedded SQL allow the application programmer greater amounts of control over the way an application interacts with the database. Although this might be desired, it also leads to greater amounts of time and resources required to develop and maintain such applications.

1.5 Call Level Interface

Another alternative to using embedded SQL within an application is to use the Call Level Interface (CLI). The difference is that, instead of placing SQL statements directly into your application code, you are calling a number of functions to perform an SQL operation. This method has a number of advantages over the use of Embedded SQL, but it may also further complicate your application.

When using the CLI, you do not need to bind your applications to the database, as is required with embedded SQL. This means than an application accessing one database may be executed against a different database without the need for recompilation or binding.

Given that there is no binding to the database, all the CLI SQL will be executed dynamically. This means that access plans and optimization will be done at run time, rather than during compilation/binding.

These features make CLI ideal for applications that will execute a large number of dynamic SQL statements across multiple databases. The application programmer will have control over data access and can check user authorizations or leave authorization to the database administrator and handle error recovery if the user has insufficient authority to perform a requested operation.

1.6 Open Database Connectivity

Microsoft's Open Database Connectivity (ODBC) is similar to the Call Level Interface in that it is a defined set of function calls that may be used to access the data contained within a database system.

CLI supports all the core, Level 1 and most of Level 2 ODBC calls. For a complete listing of the ODBC calls supported by CLI, you should refer to the *Call Level Interface Guide and Reference - for common servers*.

Many applications and database vendors support the ODBC interface; so if you are going to develop an application that will access multiple database systems, this is one of your options.

1.7 Summary

There are a number of different ways to access DB2 databases and tables. This book covers the more commonly used interfaces that application developers use when developing DB2 applications.

There are a number of tools and programming environments, such as those provided by the VisualAge and VisualGen product families, that are designed to help you build graphical interfaces into the DB2 database system.

The application development environments available are continually developing and expanding. This book is designed to give you a basic understanding of the underlying interfaces that are common to all these development environments and help you to decide which of the available interfaces is best suited to your programming needs.

Chapter 2. Command Line Processor

DB2 Version 2 provides the ability to enter database administration commands as well as SQL statements via the Command Line Processor. These commands and/or statements may be entered directly at the command line or supplied through a file.

The Command Line Processor may be used from different types of scripts, such as Bourne, Korn, C, or Perl scripts. This chapter shows the different methods that may be used to call the command line processor using these different methods.

2.1 Invoking CLP

Command Line Processor (CLP) is a utility provided by DB2 Version 2 which is an interface to the DB2 engine. You can use the CLP to execute database utilities, SQL statements and on-line help.

There are several ways to use CLP. They include:

1. Interactive Mode
2. Command Mode
3. Batch Mode

2.1.1 Interactive Mode

We can start CLP in an Interactive Mode by entering the command db2 at the operating system command prompt. This will place you into the Command Line Processor and present you with the db2⇒ prompt.

```
$ db2
(c) Copyright IBM Corporation 1993,1995
Command Line Processor for DB2 SDK 2.1.1

You can issue database manager commands and SQL statements from the
command prompt. For example:
    db2 => connect to sample
    db2 => bind sample.bnd

For general help, type: ?.
For command help, type: ? command, where command can be
the first few keywords of a database manager command. For example:
 ? CATALOG DATABASE for help on the CATALOG DATABASE command
 ? CATALOG          for help on all of the CATALOG commands.

To exit db2 Interactive Mode, type QUIT at the command prompt. Outside
interactive mode, all commands must be prefixed with 'db2'.
To list the current command option settings, type LIST COMMAND OPTIONS.

For more detailed help, refer to the On-line Reference Manual.

db2 =>
```

Figure 1. The Interactive Mode of CLP

Using this mode, you are able to write the SQL statements or any database commands and the results will be automatically formatted and displayed on the screen. This is useful for proto-typing your SQL statements or commands.

It is possible to execute an operating system command while in the Interactive Mode. To do this, you have to prefix the operating system command with an exclamation point (!) as shown in Figure 2. This capability is valid only in the UNIX and OS/2 environments.

```
db2 => ! id
uid=6(db2) gid=200(db2adm) groups=1(staff),201(db2usr),204(smadmin)
```

Figure 2. Submitting Operating System Command from Interactive Mode

To end the Interactive Mode, you should enter the `quit` command at the Command Line Processor prompt.

2.1.2 Command Mode

In this mode, you submit DB2 commands from the operating system command prompt. You need to prefix all the commands with `db2` as shown in Figure 3. You will also get the results of the command formatted as output to your screen.

```
$ db2 "select * from db2.employee"
```

Figure 3. Command Mode of CLP

This is similar to the Interactive Mode; however, you are also able to utilize any advanced features of the operating system shell that you are running in, such as the command recall facility found in many UNIX shells.

It is also necessary to escape any characters that have special meaning in the UNIX shell you are using. This may be done by either placing a backslash character () before the special character, or you may enclose the entire DB2 command or statement in double quotes.

2.1.3 Batch Mode

Instead of submitting commands by typing them one by one at the command prompt, you can also enter all the commands into a file. For example, you could execute a file, such as the one shown in Figure 4, by executing the command: `db2 -f clp.sql`

```
     list database directory
     connect to sample
     select firstnme, lastname from employee where job='CLERK'
```

Figure 4. Sample SQL Batch File, clp.sql

The results from running this file are shown in Figure 5 on page 7.

```
$ db2 -f clp.sql

 System Database Directory

 Number of entries in the directory = 2

Database 1 entry:

 Database alias                 = CELDIAL
 Database name                  = CELDIAL
 Local database directory       = /home/db2
 Database release level         = 6.00
 Comment                        =
 Directory entry type           = Indirect

Database 2 entry:

 Database alias                 = SAMPLE
 Database name                  = SAMPLE
 Local database directory       = /home/db2
 Database release level         = 6.00
 Comment                        =
 Directory entry type           = Indirect

   Database Connection Information

 Database product       = DB2/6000 2.1.1
 SQL authorization ID   = DB2
 Local database alias   = SAMPLE


FIRSTNME     LASTNAME
------------ ---------------
SEAN         O'CONNELL
JAMES        JEFFERSON
SALVATORE    MARINO
DANIEL       SMITH
SYBIL        JOHNSON
MARIA        PEREZ

  6 record(s) selected.

$
```

Figure 5. Batch Mode

As in the Interactive Mode, you can also execute an operating system command from batch mode by prefixing the operating system command with an exclamation point (!).

2.2 CLP Options

There are some options that can be set for using the Command Line Processor. Table 1 shows the options, the description and the default settings.

Table 1 (Page 1 of 2). CLP Command Options

Option Flag	Description	Default Setting
-a	To display SQLCA data	OFF

Table 1 (Page 2 of 2). CLP Command Options		
Option Flag	**Description**	**Default Setting**
-c	To automatically commit SQL statements	ON
-ecs	To display SQLCODE or SQLSTATE	OFF
-f	To read command input from a file instead of from standard input	OFF
-l	To log commands in a history file	OFF
-o	To display output data and messages to standard output	ON
-p	To display a Command Line Processor prompt when in the interactive input mode	ON
-r	To write the report generated by a command to a file	OFF
-s	To stop execution if there are errors while executing commands in a batch file or in Interactive Mode.	OFF
-t	To use a semicolon (;) as the statement termination character and disable the use of backslash (\) as a line continuation character.	OFF
-tdx	To define and to use ″x″ as the statement termination character	OFF
-v	To echo command text to standard output	OFF
-w	To display SQL statement warning message	ON
-z	To redirect all output to a file. Similar to the -r option, but -z includes any messages and error codes with the output	OFF

The following rules apply to set the options:

- -option-flag to turn on the option. For example, -c turns on the auto-commit option.
- -option-flag- or +option-flag to turn off the option. For example, -c- or +c will turn the auto-commit option off.
- Option flags are not case-sensitive; so you can either use -c or -C.
- Some options, such as -f, -l, -r, and -z, require a file name since these use a file either for input or output.

2.2.1 CLP Options Settings

There are three ways to set the command line options. They are:

1. Using DB2OPTIONS in the db2profile file

   The options that are set from the db2profile will be valid for the entire session of DB2. Figure 6 shows part of the db2profile file that sets the command line options.

```
...
DB2OPTIONS='-a +c -l'
export DB2OPTIONS
...
```

Figure 6. Set the Command Option from the db2profile File

2. Using the option flags

You can use a flag at the command line for setting the options when you are using the Command Mode or Batch Mode. In Command Mode, the options you set will be valid only for the associated command you submit to DB2. While in the Batch Mode, the options will be used for all the commands contained in the batch file.

Figure 7 shows an example of setting a command option using a flag in Command Mode. On the first and the second command, the Command Line Processor displays output data and messages to standard output. On the third command, we use the +o flag option that tells the Command Line Processor not to display output data and messages. So, even though the connection is successful, no messages are returned.

```
$ db2 connect to sample

   Database Connection Information

 Database product       = DB2/6000 2.1.1
 SQL authorization ID   = DB2
 Local database alias   = SAMPLE
$ db2 connect reset
DB20000I  The SQL command completed successfully.
$ db2 +o connect to sample
$
```

Figure 7. Setting Command Options by using Flags

3. Using the update command options command

   In the Interactive Mode, we use the update command options command to set new options. This command can also be used from within a command file. The settings will only be valid for that session. For example, to turn off the output and display the SQLCA information, you could use the command;

   ```
   db2⇒ update command options using o OFF a ON
   ```

We can see the current settings for the command options by entering the list command options command either from Interactive, Command, or Batch Mode. Figure 8 on page 10 shows an example of using this command.

```
db2 => list command options

     Command Line Processor Option Settings

 Backend process wait time (seconds)        (DB2BQTIME) = 1
 No. of retries to connect to backend        (DB2BQTRY) = 60
 Request queue wait time (seconds)          (DB2RQTIME) = 5
 Input queue wait time (seconds)            (DB2IQTIME) = 5
 Command options                           (DB2OPTIONS) =

 Option  Description                                 Current Setting
 ------  ------------------------------------------  ---------------
   -a    Display SQLCA                               OFF
   -c    Auto-Commit                                 ON
   -e    Display SQLCODE/SQLSTATE                    OFF
   -f    Read from input file                        OFF
   -l    Log commands in history file                OFF
   -o    Display output                              ON
   -p    Display interactive input prompt            ON
   -r    Save output to report file                  OFF
   -s    Stop execution on command error             OFF
   -t    Set statement termination character         OFF
   -v    Echo current command                        OFF
   -w    Display FETCH/SELECT warning messages       ON
   -z    Save all output to output file              OFF
```

Figure 8. A List of the Current Options Settings

Since there are several ways to set the CLP options, the following list indicates the order in which the final settings will be determined.

1. The CLP sets up default options.
2. The CLP reads the CLP options set by the DB2OPTIONS environment variable. This may be set in the db2profile or in the user's profile.
3. Any command line options/flags that were set will be read.
4. Finally, CLP will take input from any `update command options` command used during a session.

2.3 Using the CLP

Several things that you should consider when using the CLP in either mode are:

- The CLP commands are not case-sensitive.

  You can submit the command in either uppercase or lowercase. But there are some commands that have case-sensitive parameters. For example, the arguments for the `with` clause in the `change database comment` command does reflect the case.

- The CLP uses the character "\" as a line continuation character. It means that when the CLP finds a "\", it will read the next line and concatenate the characters on both lines. For example:

  ```
  db2 => select empno, firstnme, midinit, lastname \
  db2 (cont.) => from employee \
  db2 (cont.) => where job='CLERK'
  ```

- Special characters, such as $, &, *, (,), ;, <, >, ?, ', and ", can be used within CLP commands only if you are using the Interactive or Batch Modes.

If you use these characters in Command Mode, which is an operating system shell, they may return a syntax error message. This is because the shell may try to interpret these characters. There are two ways available to overcome this problem:

1. Put the statement between quotation marks. For example:

```
$ db2 "select * from employee"
```

2. Use an escape character, that is "\", before the special character. For example:

```
$ db2 select \* from employee
```

- The CLP is not a programming language; so you cannot use any host variables in your commands.

2.4 Using CLP in an Application

You can use the CLP from many types of UNIX scripts to develop an application. You could use the C, Korn, or Bourne shell, depending on which script you prefer.

When you run any CLP commands, a code is returned which indicates success or failure of the CLP operation. This code is returned to the calling shell or application. Table 2 lists the possible return codes and their descriptions.

Table 2. CLP Return Codes

Code	Description
0	DB2 command or SQL statement executed successfully
1	SELECT or FETCH statement returned no rows
2	DB2 command or SQL statement warning
4	DB2 command or SQL statement error
8	Command Line Processor system error

Figure 9 shows a Bourne script that makes use of the return code from the `get database manager configuration` command.

```
...
db2 get database manager configuration
if ["$?" = "0"]
   then echo "OK!"
fi
...
```

Figure 9. Using the Return Code in an Application

The Stop Execution on Command Error option (-s option) affects the execution of a CLP command depending on the Return Code that resulted from the command. This effect occurs when the commands are submitted in the Interactive Mode or from an input file in the Batch Mode. Table 3 on page 12 summarizes how the -s option affects the execution of a CLP command.

Table 3. CLP Return Codes and Command Execution

Return Code	**-s Option Set**	**+s Option Set**
0 (success)	execution continues	execution continues
1 (no rows selected)	execution continues	execution continues
2 (warning)	execution continues	execution continues
4 (DB2 or SQL error)	execution stops	execution continues
8 (System error)	execution stops	execution stops

Chapter 3. World Wide Web

DB2 databases can be accessed through any WWW browser that supports Version 2.0 of the HyperText Markup Language (HTML). This chapter discusses the DB2 World Wide Web Connection (DB2 WWW Connection) product and the different programming languages that enable developers to create Web applications for accessing DB2 databases. The creation and implementation of such applications is also discussed here.

3.1 Overview

Internet users are growing in number. More and more people access the World Wide Web (WWW) to find information they need. These people can be your customers, your employees, your colleagues, or anyone who needs information from your company that is stored on your DB2 databases.

With the DB2 WWW Connection product, it is possible to make your DB2 databases available to a user with any of the WWW browsers that are available today. Access to the DB2 databases is also possible by using Hypertext Markup Language (HTML) and Common Gateway Interface (CGI) scripts. However, the DB2 WWW Connection product provides a simpler and more complete interface into the DB2 databases, with higher levels of functionality than could be achieved through HTML and CGI scripts alone.

3.1.1 DB2 World Wide Web Connection

DB2 WWW Connection V1 is a product that allows users to create WWW applications that can access DB2 databases. It has the ability to pass variables from HTML to SQL and back. And you don't need to make any change in the DB2 database structure.

Application developers who wish to access data in DB2 databases only have to write Structured Query Language (SQL) statements in *macro files* or *CGI scripts* and combine them with Hypertext Markup Language (HTML) to produce their applications.

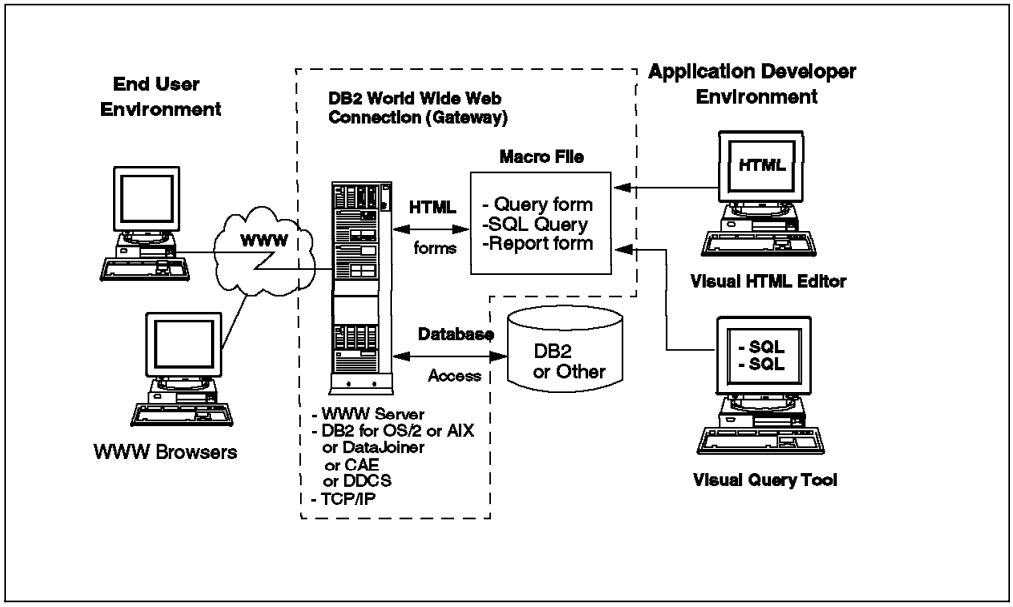


Figure 10. DB2 World Wide Web Connection

3.1.2 Client/Server Environment

DB2 WWW Connection can be used in two client/server environments:

- Two-tier

  A two-tier client/server is an environment with a local WWW server on the same machine as the DB2 server. It can have one or more Web browser clients.

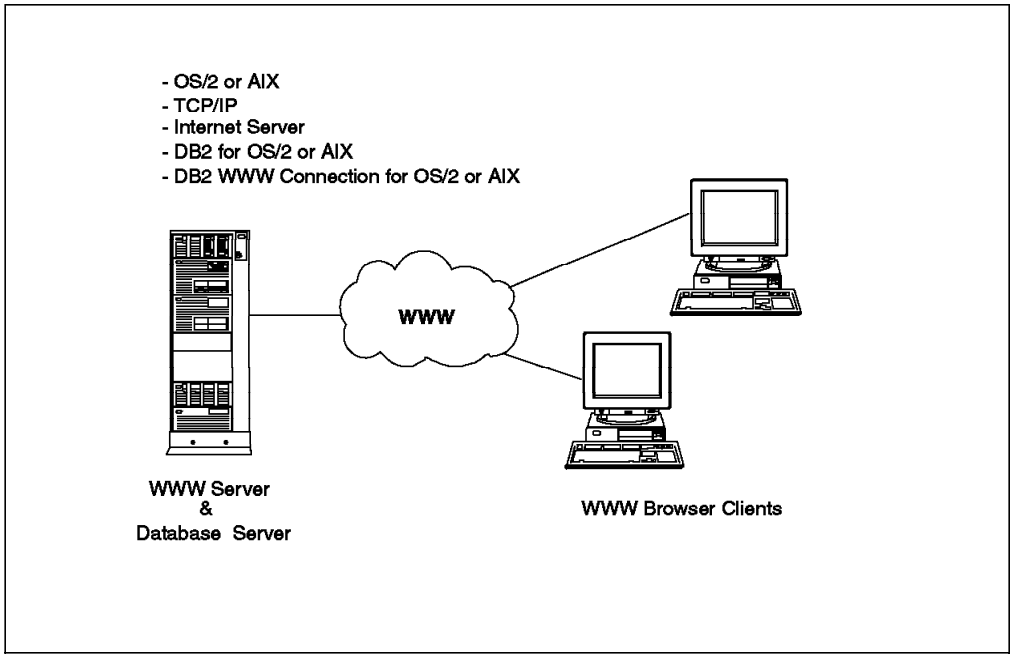


Figure 11. A Two-Ttier Client/Server Environment

- Three-tier

  A three-tier client/server is an environment where a local WWW server with one or more WWW browser clients accesses remote DB2 servers.

Your WWW application can access DB2 data on the local WWW server as well as data on remote database servers connected with Distributed Database Connection Services (DDCS), Client Application Enabler (CAE) or DataJoiner. Using DataJoiner, your WWW application can even access Oracle, SYBASE or other relational and non-relational data sources.

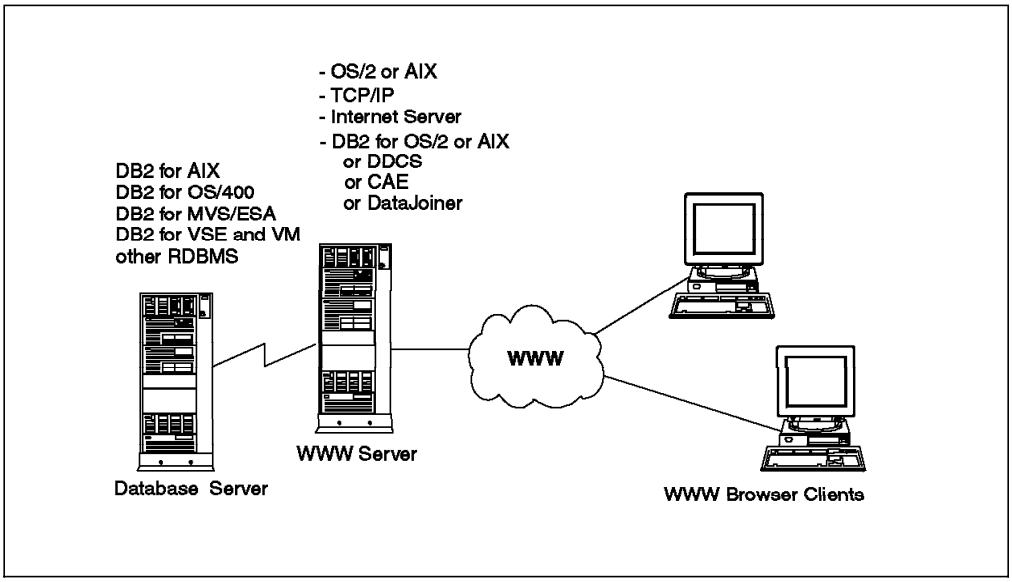


Figure 12. A Three-tier Client/Server Environment

3.2 DB2 WWW Connection Installation

DB2 WWW Connection distribution media contains the following files:

README.AIX Documentation on how to install and set up DB2 WWW Connection

db2www.pkg Installation configuration file

license International Program License Agreement

The product will be installed in the default directory, /usr/lpp/internet.

To install DB2 WWW Connection, you can use the System Management Interface Tool (SMIT).

1. Make sure that you are logged on as *root* on the server where you wish to install the product.

   Check it by submitting the following commands:

   `hostname`; the result should be your server name.

   `whoami`; the result should be `root`.

2. Type the following to begin: `smit` or `smitty`
3. You will now see the System Management screen. Place the cursor on Software Installation and Maintenance, and press **Enter**.

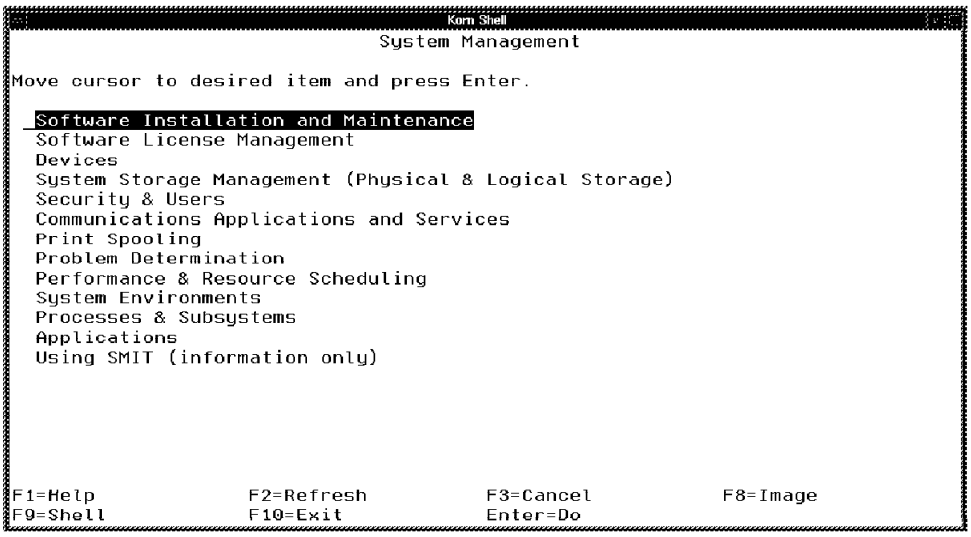


Figure 13. System Management Interface Tool Screen

4. Choose **Install and Update Software** on the Software Installation and Maintenance screen.

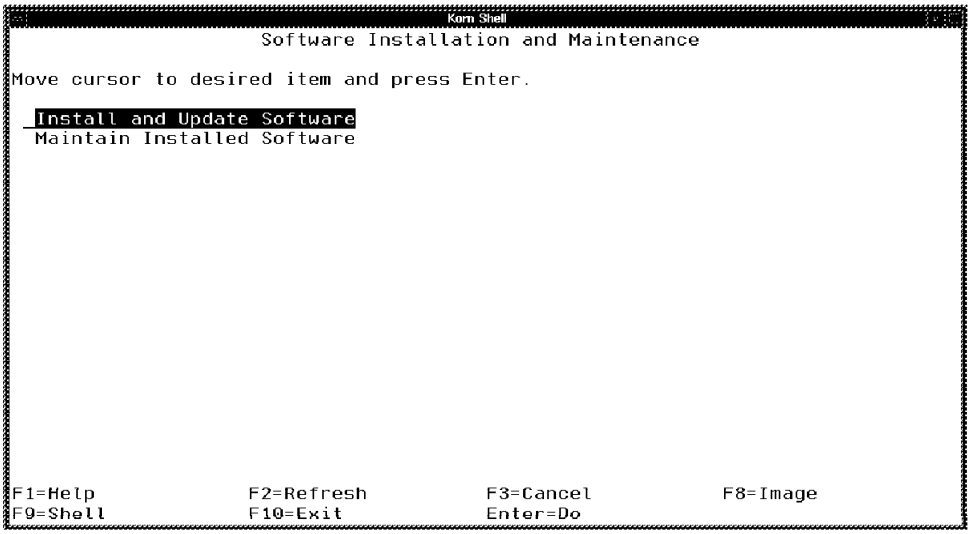


Figure 14. Software Installation and Maintenance Screen

5. Choose **Install/Update Selectable Software (Custom Install)** on the Install and Update Software screen.

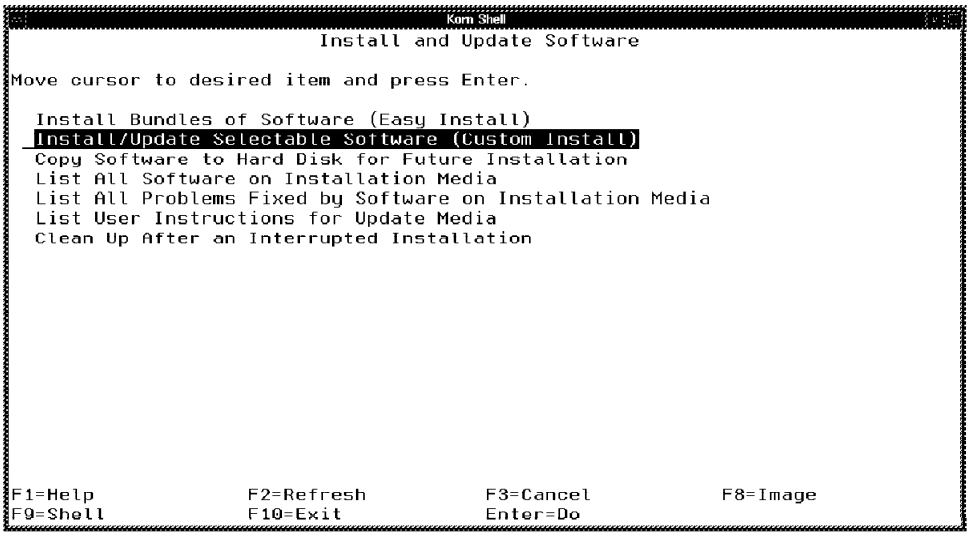


Figure 15. Install and Update Software Screen

6. Choose **Install Software Products at Latest Level** on the Install/Update Selectable Software (Custom Install) screen.

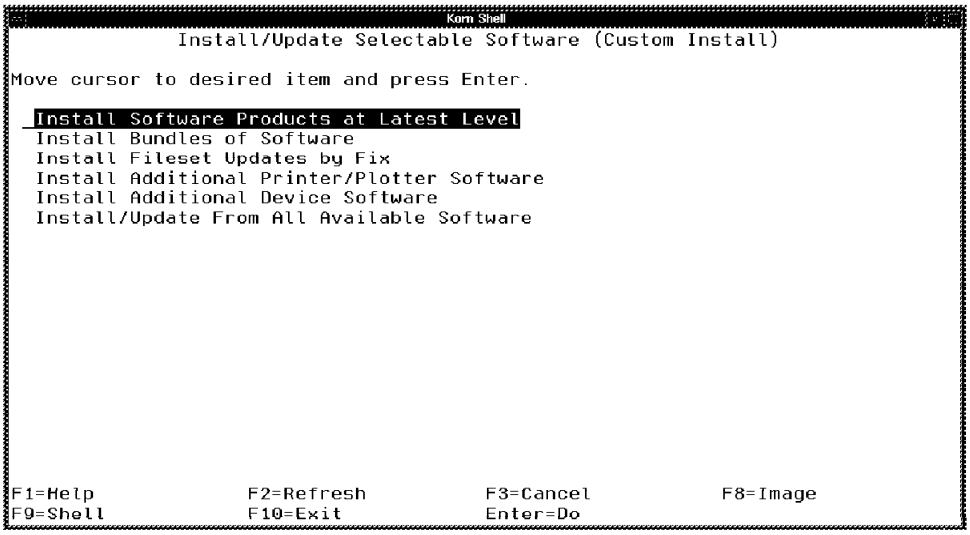


Figure 16. Install/Update Selectable Software (Custom Install) Screen

7. Choose **Install New Software Product at Latest Level** on the Install Software Products at Latest Level screen.

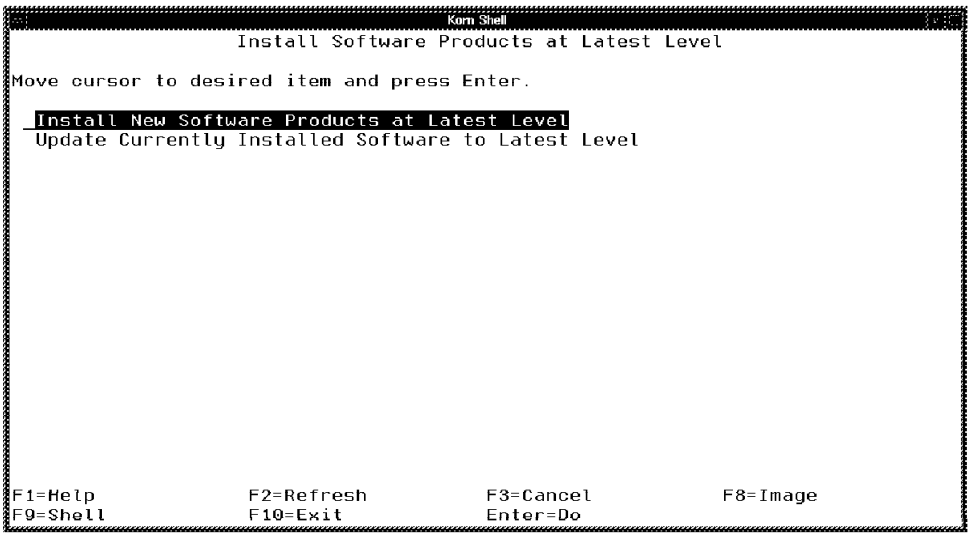


Figure 17. Install Software Products at Latest Level Screen

8. Now, as in Figure 18, type your input device name on the entry field, and press **Enter**. If you are not sure what devices are on your system, press **F4** to see a list of devices.

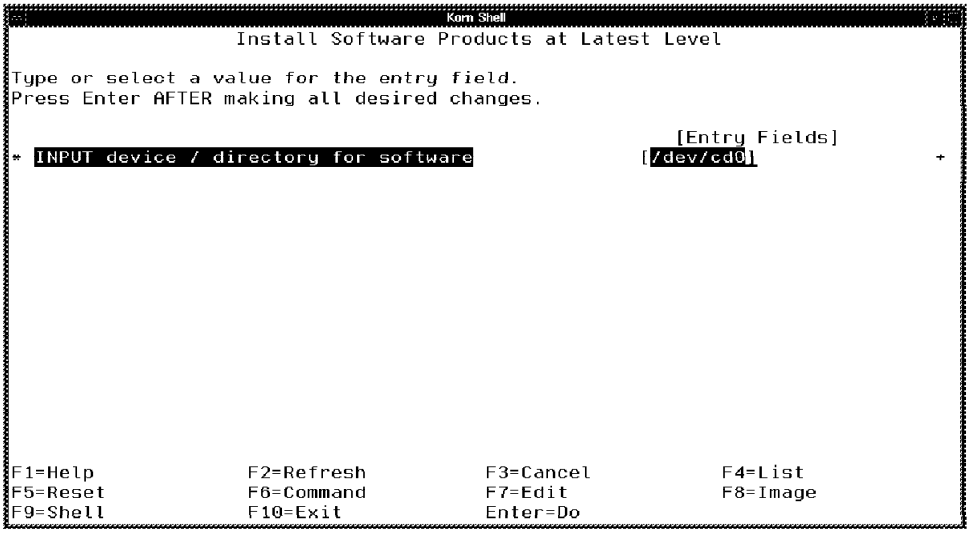


Figure 18. Install Software Products at Latest Level Screen

9. Position the cursor on SOFTWARE to install, and press **F4**.

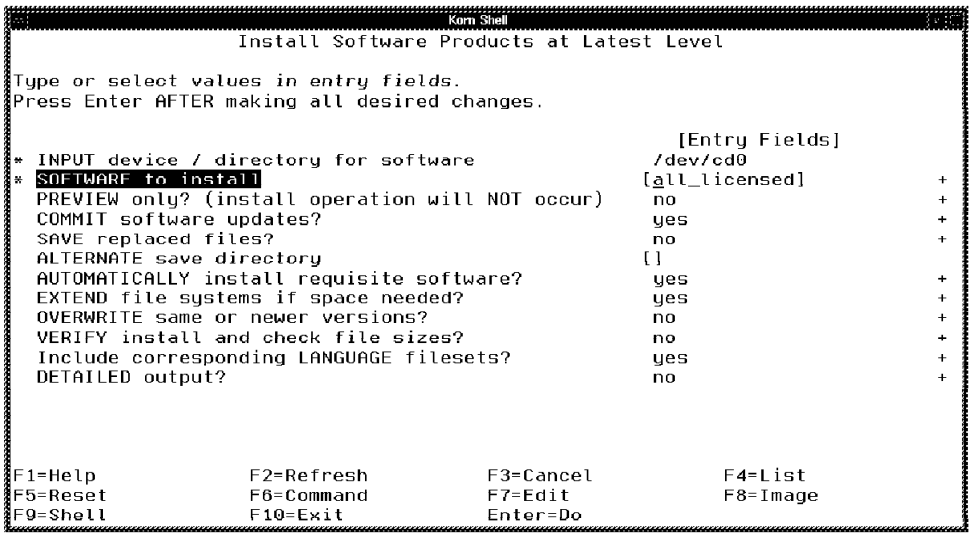


Figure 19. Install Software Products at Latest Level Screen

10. You will see a list of software to install. Select either:

    DB2 WWW Connection for DB2 Version 1 if you have DB2/6000 Version 1 or 1.2, or

    DB2 WWW Connection for DB2 Version 2 if you have DB2 for AIX Version 2 or higher.

    To make a selection, position the cursor on the line you will choose, and press **F7**.

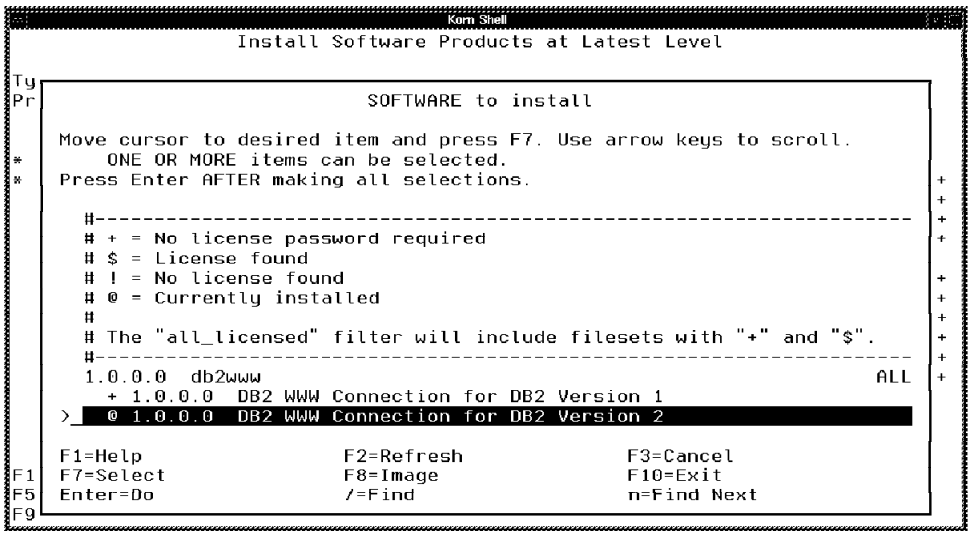


Figure 20. SOFTWARE to Install Screen

11. Press **Enter** to install the product you choose.

3.3 Setting Up DB2 WWW Connection

This section describes the procedures needed to set up the DB2 WWW Connection product. We assume that the World Wide Web/Internet server is already configured and running on your system.

3.3.1 Internet Server Parameters

Before you continue, you will need to know the following environment information about your existing WWW server's configuration.

- WWW server directory
- WWW server document directory
- WWW server image file directory
- WWW server CGI executable programs directory
- DB2 instance name

Table 4 shows the Internet server parameters defined for this example.

Table 4. An Example of WWW/Internet Server Parameters

Parameter Description	**Parameter Value**
WWW Server directory	/usr/local/www
WWW Server Document directory	/usr/local/www/pub
WWW Server Image file directory	/usr/local/www/icons
WWW Server CGI executable programs directory	/usr/local/www/cgi-bin
DB2 Instance Name	db2

In section 3.3.2, "DB2 WWW Connection Configuration," we will use the values shown in Table 4.

3.3.2 DB2 WWW Connection Configuration

By default, your WWW server directory is set to /usr/lpp/internet/server_root if you are using the IBM Internet Connection Server product. You can determine what your WWW server directory is by looking at the WWW server configuration file, which is normally /etc/httpd.conf. There, you will find the *ServerRoot* parameter which defines your WWW server's root directory.

```
vi /etc/httpd.conf
```

```
# @(#)53  1.9 src/web/etc/httpd.conf, web, web41C, 9535B 6/21/95 10:19:08
#
# COMPONENT_NAME: web httpd.conf
#
# FUNCTIONS:
#
# ORIGINS: 10  26  27
#
# (C) COPYRIGHT International Business Machines Corp. 1995
# All Rights Reserved
# Licensed Materials - Property of IBM
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
##############################################################################
#
#       Sample configuration file for httpd for running it
#       as a normal HTTP server.
#
# See:
# <file://localhost/usr/lpp/internet/server_root/pub/ServerDocs/
#         index.html>
#
##############################################################################

#
#       Set this to point to the directory where you unpacked this
#       distribution, or wherever you want httpd to have its "home"
#
ServerRoot     /usr/lpp/internet/server_root

#
#       The default port for HTTP is 80; if you are not root you have
#       to use a port above 1024; good defaults are 8000, 8001, 8080
#
Port     80

#
#
UserId     nobody
GroupId    nobody
```

Figure 21. Part of the WWW Server Configuration File

Figure 21 shows part of the /etc/httpd.conf file. The documentation with the IBM Internet Connection Server product contains sample configuration files for different types of server environments.

If your WWW server directory is not the default, /usr/lpp/internet/server_root, then you will need to copy the following files to the new location. For this example, the server root directory is /usr/local/www.

1. Copy db2www and db2sql.bnd files to your WWW server CGI executable program directory.

   ```
   cp /usr/lpp/internet/server_root/cgi-bin/db2www  \
      /usr/local/www/cgi-bin
   ```

   ```
   cp /usr/lpp/internet/server_root/cgi-bin/db2sql.bnd  \
      /usr/local/www/cgi-bin
   ```

2. Copy db2www.ini and all *htm* files to your WWW server document root directory.

```
cp /usr/lpp/internet/server_root/pub/db2www.ini  \
   /usr/local/www/pub

cp /usr/lpp/internet/server_root/pub/*.htm  \
   /usr/local/www/pub
```

3. Copy all icon files to your WWW server image directory.

```
cp /usr/lpp/internet/server_root/icons/*  \
   /usr/local/www/icons
```

4. Create a tmplobs directory under your WWW server document directory, and make sure that your WWW server has the write access to this directory. The least access is 770.

```
mkdir /usr/local/www/pub/tmplobs

chmod 770 /usr/local/www/pub/tmplobs
```

   So, if your WWW server was started by user ID *nobody* and groupid *nobody* (as defined in the /etc/httpd.conf file), the access list of the tmplobs directory should be at least:

```
drwxrwx---   2 nobody   nobody      2560 Jan 25 14:11 tmplobs
```

Figure 22. Access List of tmplobs Directory

5. Create a macro directory under your WWW server CGI executable programs directory, and copy all .d2w files to this directory.

```
mkdir /usr/local/www/cgi-bin/MACRO

cp /usr/lpp/internet/db2www/macro/*.d2w \
   /usr/local/www/cgi-bin/MACRO
```

The file db2www.ini is the configuration file for DB2 WWW Connection. It contains three variables that are described as the following:

MACRO PATH Specifies the directory where the macro files exist

BINDFILE Specifies the location of the bind file for DB2 WWW Connection

DB2INSTANCE Specifies the DB2 instance name

Figure 23 contains a sample of the db2www.ini file.

```
MACRO_PATH /usr/local/www/cgi-bin/MACRO
BINDFILE /usr/local/www/cgi-bin/db2sql.bnd
DB2INSTANCE db2
```

Figure 23. Sample of the db2www.ini File

3.4 Application Development

Many people use the Hypertext Markup Language (HTML) to create WWW applications. HTML files can be written using any text editor, and they consist of markup tags used to tell the WWW browser how to display text or images.

An application created using HTML is a static application which will always display the same information until the source documents have been changed.

To access data in any database, we need an application that can run SQL statements dynamically. There are two ways to do so:

1. Using Macro Files

   Using macro files, you are able to format the output from SQL queries and reports by using HTML and macro variables that contain the data returned by the SQL statement. The complete SQL command is dynamically built with the user input and then sent to the database. The query results are shown in a report form defined by the HTML markup.

2. Using CGI Scripts

   CGI script is another way to develop a dynamic WWW application. These applications are dynamic because the results may change each time the script is executed. The script itself remains the same.

   CGI script can be written using any language that the system supports, such as C/C++, Fortran or any UNIX shell.

Table 5 summarizes some characteristics of the two ways to access databases when using DB2 WWW Connection.

Table 5 (Page 1 of 2). Using Macro Files and CGI Scripts to Access Databases

	CGI Scripts	**Macro Files**
Supported Languages	• Any UNIX shell • C/C++ • COBOL • Fortran • PASCAL • ADA • Perl • Any programming language supported by the platform	• Macro file commands
Other commands included	• SQL statements	• SQL statements • HTML commands
Define Section	• Input variables	• Input and Output variables
HTML Input Section	• In other program	• Imbedded
SQL Section	• Dynamic • Result as a file	• Dynamic • Result as column and row variables
HTML Output Section	• Imbedded • Print the result	• Imbedded • HTML form as a report

Table 5 (Page 2 of 2). Using Macro Files and CGI Scripts to Access Databases		
	CGI Scripts	**Macro Files**

Execution	• CGI from HTML:
	`<FORM METHOD=POST`
	`ACTION="/cgi-bin/{CGI Script}">`
	• CGI from URL:
	`http://{WWW server}/cgi-bin/{CGI Script}`
		• Macro file from HTML:
		`<FORM METHOD=POST`
		`ACTION="/cgi-bin/db2www/{macro file}/{cmd}">`
		• Macro file from URL:
		`http://{WWW server}/cgi-bin/db2www/{macro file}/{cmd}`

3.4.1 Macro Files

Macro files are written by application developers to create WWW applications. All macro files should be located in a macro directory on the WWW server. This macro directory is usually a subdirectory of the CGI directory.

Macro files contain sections that are written using Hypertext Markup Language (HTML) and Structured Query Language (SQL).

These sections are:

1. DEFINE Section

   This is used to define input and output variables in a macro file. Since the application will access data on a database, you should define at least the DATABASE variable.

2. SQL Section

   You will use this section to write an SQL query. You can have many SQL sections, but each one can only contain one SQL query. Also, you can have SQL_REPORT and SQL_MESSAGE subsections.

3. HTML Input Section

   In this section, you specify the HTML page used for the input of values needed to build the SQL query.

4. HTML Report Section

   This section allows you to specify the format to be used for showing the query results. This section will execute SQL queries defined in each SQL section.

5. Comment Section

   The comment section allows you to write a description of the macro file. This is recommended, but is optional.

You can find a macro file template in the directory:

```
/usr/lpp/internet/db2www/macro
```

There are two ways you can invoke macro files in HTML documents:

- Using an Anchor reference

  <a ref=http://{WebServer}/cgi-bin/db2www/{macro file}/{cmd}{?}

- Using an HTML form

  <form method={method}
  action=http://{WebServer}/cgi-bin/db2www/{macro file}/{cmd}{?}>

Where:

{*WWW Server*} is your WWW server address.

{*macro file*} is any macro file written with a .d2w. This file should be saved in your macro directory.

{*cmd*} can be *input* or *report* to indicate if the macro is to execute as an input form or execute the report section.

{*method*} The method determines the way input data is sent to the CGI or macro. Two methods, get and post, are available. Input using the get method is appended after the *?* in the URL and is limited to 256 characters. As this is a part of the URL, information such as the user ID and password will be display in the URL location field and stored in the WWW server's log files. This can lead to serious security violations. The other method, post, sends URL information through standard input, which is not limited in size and does not suffer the security problems associated with the *get* method.

3.4.2 Handling Large Objects (LOBs)

DB2 WWW Connection supports BLOBs, CLOBs and DBCLOBs data types to be queried from DB2 V2.1 databases. You have to consider limitations such as resources, special hardware and software requirements. Using LOBs can quickly consume resources because it is usually a big file. Some LOBs, such as an audio file, require special hardware and software installed on your machine in order to run.

LOBs resulting from an SQL query in a macro file will be saved in a file under a temporary directory named tmplobs. This directory is located under the WWW server's root directory.

The first section of a LOB field in the DB2 WWW macro contains a file signature of its type; when DB2 WWW Connection recognizes the LOB, the extension is added to the temporary file so it can be displayed. If not, the temporary file must be renamed with the appropriate extension (See 3.6.3, "Using LOBs in a Macro File" on page 35).

Character Large Objects (CLOBs) are assigned a .txt extensions. For Binary Large Objects (BLOBs), bitmaps are recognized with a .bmp extensions, graphical image formats with .gif extensions, and tag image file formats with .tif extensions. Postscript is only identified when contained in BLOBs, not in CLOBs.

Other types are not recognized, and no extension is added.

3.5 Security

Security is one of the most important issues. We need to be sure that we share our data only with our intended audience. We can have different levels of security on DB2 WWW Connection applications:

- Authentication

  Configure your WWW server configuration file, /etc/httpd.conf, to protect certain directories on your WWW server, including your DB2 server directories.

Use LOGIN and PASSWORD variables on the DB2 WWW Connection macro files to restrict access to any users. These variables are passed to DB2 for authentication at the table and column level.

- Encryption

  Secured Sockets Layer (SSL) or Secured Hypertext Transfer Protocol (SHTTP) products can be used to protect authenticated user IDs/passwords and any document transferred.

- Firewall

  IBM's NetSP Firewall and other products can also be used.

3.5.1 Macro Files Without LOGIN and PASSWORD Variables

As a relational database, DB2 has an authentication mechanism that uses user ID and password logins. We can use this authentication mechanism in DB2 WWW Connection applications.

Usually, there are *user ID* and *password* parameters in the WWW server configuration file, /etc/httpd.conf, with *nobody* specified for use as default values (see Figure 21 on page 21). When you run an SQL query to access a DB2 database using a DB2 WWW Connection macro file, you will use the user ID and group specified in the /etc/httpd.conf file as the default connection user ID and table schema.

This condition might result in some errors with your macro file if you have not taken this into account. Some typical errors include:

1. SQL0204N **"**NOBODY.tablename**"** is an undefined name

   You have to use valid qualifiers to define your SQL query statement; otherwise NOBODY will be used as the default qualifier.

2. SQL0551N **"**NOBODY**"** does not have the privilege to perform operation **"**SELECT**"** on object **"**tablename**"**

   You have to grant select privilege to user nobody or any user who intends to execute this query. This is because the authentication of users is done at the time of execution.

You should never use a system administration user ID or the instance owner's user ID as the default user ID or group ID.

3.5.2 Macro Files With LOGIN and PASSWORD Variables

DB2 WWW Connection has some special variables. Two of them are used to apply the DB2 authentication mechanism:

LOGIN specifies the user ID which accesses DB2 databases

PASSWORD specifies the password associated with the LOGIN variable

If these two variables are used in the macro file, they will override any value on the user Id parameter in the WWW server configuration file, /etc/httpd.conf.

You will use LOGIN and PASSWORD variables in your macro file if you want to share your data only with certain people. Do the following to avoid possible errors.

- Use specific qualifiers for your tables in the macro file to avoid the SQL query being executed using LOGIN variables.

- Revoke SELECT privilege from public and user NOBODY. This will forbid people who do not have access to data to run the query.

3.6 Examples

The following samples show you how to easily change your Korn shell scripts or write new ones in WWW browser presentations.

Figure 24 lists a Korn shell script that is intended to receive input data from the user and then build a query that will execute against a DB2 database. This script should have execute permission so that it may be used.

```
#!usr/bin/ksh

# Prompt for database name
#
echo ""
echo "Enter database name : \c"; read dbname
if [ "$dbname" != "" ]
then
   # Prompt for User and Password
   #
   echo "Enter User Name : \c"; read user
   if [ "$user" != "" ]
   then
       db2 connect to $dbname user $user
   else
       db2 connect to $dbname
   fi
fi

# Prompt for Table Name and/or Owner
#
echo ""
echo "Enter Table Name   : \c"; read tname
if [ "$tname" = "" ]; then tname='%'; fi

echo "Enter Table Owner  : \c"; read oname
if [ "$oname" = "" ]; then oname='%'; fi

# Run Query
#
db2 "select \
     tabname TABLE, tabschema SCHEMA, colname COLUMN, \
     typename TYPE, length LENGTH \
     from syscat.columns \
     where tabname like UCASE('$tname%') and \
           tabschema like UCASE('$oname%')"
```

Figure 24. Access DB2 Databases from Korn Shell Scripts: dbcols

The results from executing this script can be seen in Figure 25 on page 28.

```
Enter Database Name : sample
Enter User Name     : guest
Enter password for guest:

   Database Connection Information

 Database product       = DB2/6000 2.1.1
 SQL authorization ID   = GUEST
 Local database alias   = SAMPLE

Enter Table Name    : employee
Enter Table Owner   : db2

TABLE          SCHEMA   COLUMN            TYPE                LENGTH
-------------- -------- ----------------- ------------------ -----------
EMPLOYEE       DB2      BIRTHDATE         DATE                         4
EMPLOYEE       DB2      BONUS             DECIMAL                      9
EMPLOYEE       DB2      COMM              DECIMAL                      9
EMPLOYEE       DB2      EDLEVEL           SMALLINT                     2
EMPLOYEE       DB2      EMPNO             CHARACTER                    6
EMPLOYEE       DB2      FIRSTNME          VARCHAR                     12
EMPLOYEE       DB2      HIREDATE          DATE                         4
EMPLOYEE       DB2      JOB               CHARACTER                    8
EMPLOYEE       DB2      LASTNAME          VARCHAR                     15
EMPLOYEE       DB2      MIDINIT           CHARACTER                    1
EMPLOYEE       DB2      PHONENO           CHARACTER                    4
EMPLOYEE       DB2      SALARY            DECIMAL                      9
EMPLOYEE       DB2      SEX               CHARACTER                    1
EMPLOYEE       DB2      WORKDEPT          CHARACTER                    3

  14 record(s) selected.
```

Figure 25. Sample Output from the dbcols Shell Script

3.6.1 CGI Scripts Example

The following sample shows you how to call a CGI script from an HTML file. Using both files, you can access data in the DB2 databases from any WWW browser.

To test this sample, write and save the lsdbcols.html file, shown in Figure 26 on page 29, to your WWW server document directory. Also, write and save lsdbcols.pp, shown in Figure 27 on page 30, in your WWW server CGI executable programs directory; then input the following:

```
http://{WWW server}/lsdbcols.html
```

This is the HTML file that defines the input form and calls the CGI script to run a query.

```
<HTML>          1

<TITLE>DATABASE 2 for AIX, List Columns</TITLE>

<CENTER><H1>List Columns for Table(s)</H1></CENTER>

<HR>
<P>Complete the following. If no database is selected then
the "SAMPLE" database will be used.

<FORM METHOD=POST ACTION="/cgi-bin/lsdbcols">          2

<PRE>                                          3
      Database Name_: <INPUT TYPE=TEXT NAME=dbname VALUE=SAMPLE LENGTH=8></INPUT>

      User Name_____: <INPUT TYPE=TEXT     NAME=user   LENGTH=8></INPUT>
      User Password_: <INPUT TYPE=PASSWORD NAME=passwd LENGTH=8></INPUT>

<B>NOTE: Enter both User Name and Password or leave both fields blank.</B>

      Owner/Schema__: <INPUT TYPE=TEXT NAME=oname  LENGTH=8></INPUT>
      Table Name____: <INPUT TYPE=TEXT NAME=tname  LENGTH=8></INPUT>

      <INPUT TYPE=SUBMIT VALUE="Run Query">  <INPUT TYPE=RESET VALUE="Reset Query">
</PRE>
</FORM>
</BODY>
<!------------------------------------------------------------------------>
<CENTER><AUTHOR>                    4
<P>Any comments should be sent to <A HREF=mailto:rusconi@austin.ibm.com>
Frank Rusconi</A>.

<P>Last Update:
Fri Jan 19 16:13:54 CST 1996
</AUTHOR></CENTER>

</HTML>
```

Figure 26. Access DB2 Databases from WWW Browsers Using CGI Scripts: lsdbcols.html

1 You have to define the title and headers for the form window.

2 Select the method you wish to use for the URL, post or get. Write the CGI script file name that will be executed; remember that this file must be in the cgi-bin directory and made executable with 755 mode.

3 This section contains all the input fields. These can be text fields, list boxes, select buttons, or any other definition that HTML allows.

4 You may also add some references to be linked directly from your form; these may include other HTML documents or executable programs.

Figure 27 on page 30 lists the CGI file called from lsdbcols.html, which is used to run the SQL queries.

```
#!/usr/bin/ksh        1

# Set up DB2 environment
. ~db2/sqllib/db2profile
# Read in options
read INPUT_LINE

echo "Content-type: text/html"        2
echo ""
echo "<CENTER><H1>DB2 Query - Results</H1></CENTER>"
echo "<HR>"
echo "<PRE>"        3

# Prompt for Database
#
dbname=echo $INPUT_LINE | /usr/bin/awk -F'&' '{ print $1 }'
dbname=echo $dbname | cut -d= -f2
if [ "$dbname" = "" ];then dbname="sample"; fi

# Prompt for User and Password
#
user=echo $INPUT_LINE | /usr/bin/awk -F'&' '{ print $2 }'
user=echo $user | cut -d= -f2
if [ "$user" != "" ]
then
        passwd=echo $INPUT_LINE | /usr/bin/awk -F'&' '{ print $3 }'
        passwd=echo $passwd | cut -d= -f2
        db2 connect to $dbname user $user using $passwd
else
        db2 connect to $dbname
fi

# Prompt to Table Name and/or Owner
#
oname=echo $INPUT_LINE | /usr/bin/awk -F'&' '{ print $4 }'
oname=echo $oname | cut -d= -f2
oname="$oname%"

tname=echo $INPUT_LINE | /usr/bin/awk -F'&' '{ print $5 }'
tname=echo $tname | cut -d= -f2 | tr -d '\r'
tname="$tname%"

# Run Query       4
#
db2 "select tabname TABLE, tabschema SCHEMA, colname COLUMN,
            typename TYPE, length LENGTH from syscat.columns
     where  tabname like UCASE('$tname')  and
            tabschema like UCASE('$oname')"

echo "</PRE>"
exit 0
```

Figure 27. Access DB2 Databases from WWW Browsers Using CGI Scripts: lsdbcols.pp

1 In this section, you set up the environment. You can run the db2profile (include its path) to set up default instance, user ID, password, and command options. Other shell environment variables could be set.

2 To define title and headers for the query results window, you can use HTML markup.

3 You must read all the input text fields into variables that will be used to define the query statement.

4 Finally, you enter the SQL query command. It may contain references to some or all of the variables you read before.

Figure 28 shows what the HTML screen will look like when you run the CGI script sample above.

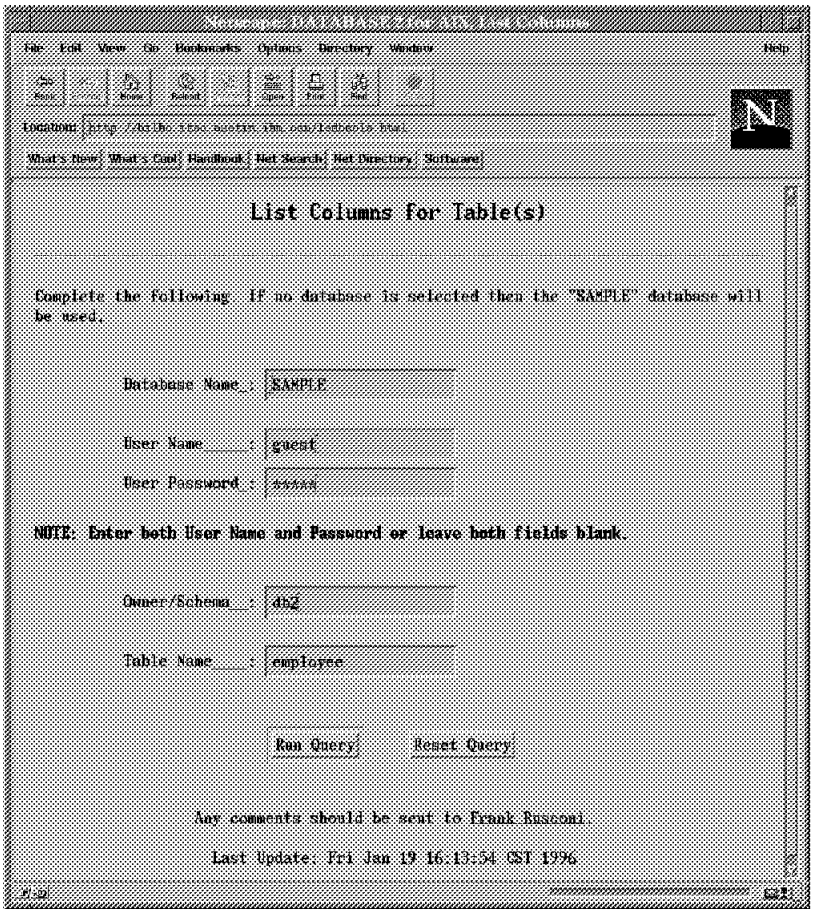


Figure 28. HTML Input Form Used By lsdbcols.html

The output of the query running on lsdbcols.pp is shown in Figure 29 on page 32.

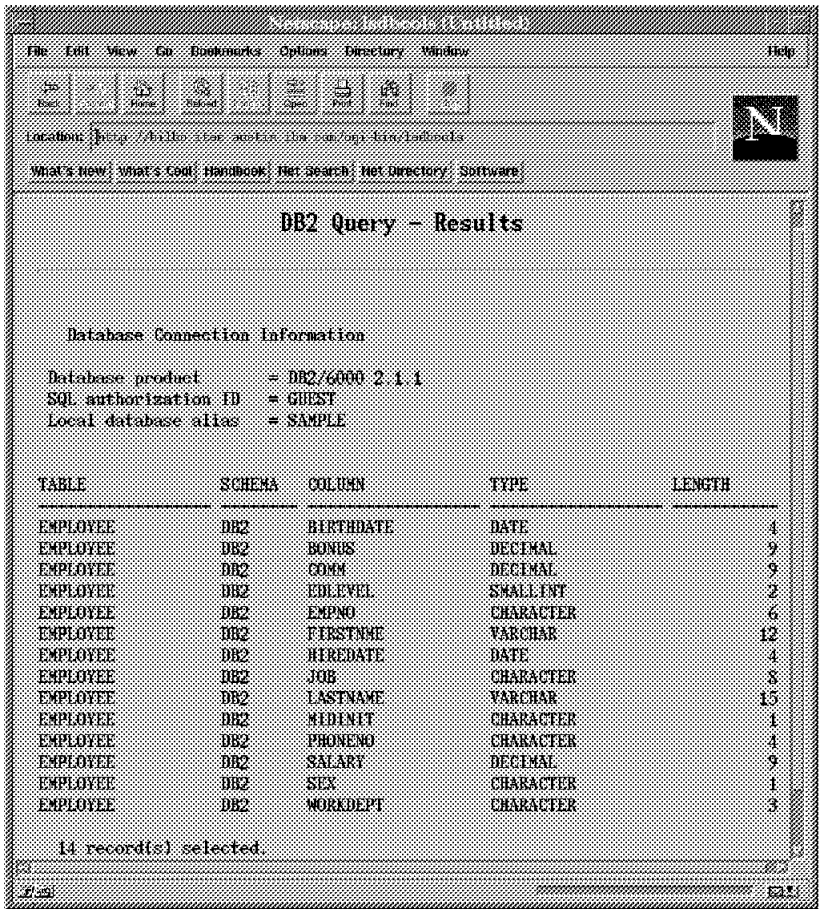


Figure 29. Output from CGI Script

3.6.2 DB2 WWW Macro Example

This example shows the input form that requires you to enter values that will then be used to perform an SQL query. Once you submit the query, the results will be returned as a HTML document.

To run this macro file, write and save it under your WWW macro directory, and input the following in the URL:

```
http://{WWW server}/cgi-bin/db2www/lsdbcols.d2w/input
```

```
%HTML_INPUT{
<HTML>
<TITLE>DATABASE 2 for AIX, List Columns</TITLE>  1
<CENTER><H1>List Columns for Table(s)</H1></CENTER>
<HR>
<P>Complete the following. If no database is selected then the "SAMPLE"
database will be used.

<FORM METHOD=POST ACTION="/cgi-bin/db2www/lsdbcols.d2w/report"> 2

<PRE>                                                              3
     Database Name_: <INPUT TYPE=TEXT NAME=DATABASE VALUE=SAMPLE  LENGTH=8></INPUT>

     User Name_____: <INPUT TYPE=TEXT     NAME=LOGIN    LENGTH=8></INPUT>
     User Password_: <INPUT TYPE=PASSWORD NAME=PASSWORD LENGTH=8></INPUT>

<B>NOTE: Enter both User Name and Password or leave both fields blank.</B>

     Owner/Schema__: <INPUT TYPE=TEXT NAME=oname  LENGTH=8></INPUT>
     Table Name____: <INPUT TYPE=TEXT NAME=tname  LENGTH=18></INPUT>

       <INPUT TYPE=SUBMIT VALUE="Run Query"> <INPUT TYPE=RESET VALUE="Reset Query">
                                                                   4
</PRE>
</FORM>
</BODY>
<!-------------------------------------------------------------------------->
<CENTER><AUTHOR>   5
<P>Any comments should be sent to <A HREF=mailto:rusconi@austin.ibm.com>
Frank Rusconi</A>.

<P>Last Update:
Wed Jan 24 09:38:32 CST 1996
</AUTHOR></CENTER>
</HTML>
%}

%SQL{   6
SELECT tabname TABLE, tabschema SCHEMA, colname COLUMN,
       typename TYPE, length LENGTH
FROM   syscat.columns
WHERE  tabschema like ucase('$(oname)%')  AND
       tabname   like ucase('$(tname)%')

%SQL_REPORT{   7
<H2>SQL Report Results</H2>
<TABLE BORDER WIDTH="100%">
<TH>$(N1)<TH>$(N2)<TH>$(N3)<TH>$(N4)<TH>$(N5)
%ROW{   8
<TR><TD>$(V1)<TD>$(V2)<TD>$(V3)<TD>$(V4)<TD>$(V5)
%}
</TABLE>
%}
%}

%HTML_REPORT{   9
%EXEC_SQL
%}
```

Figure 30. Sample DB2 WWW Macro: lsdbcols.d2w

1 This line defines the header of the WWW page used in the application. All is written using standard HTML.

2 This line calls the macro file `lsdbcols.d2w`, for a report form. We are using the POST method because it's safer than GET (See 3.4.1, "Macro Files" on page 24).

3 This part is used to input the values needed to build a query. We are using LOGIN and PASSWORD variables to pass them to DB2. If a user leaves them blank, DB2 will take the user ID specified in the WWW server configuration file as a user ID login to DB2.

4 This line will show two buttons on the WWW page screen. One has “Run Query” text on it and the other has “Reset Query” text. If a user clicks on the Run Query button, the query will be submitted to DB2. And if a user clicks on Reset Query, the value of the variables will be set to default values.

5 In this part, we write any text as comments or reference.

All the above parts, **1**, **2**, **3**, **4** and **5**, create the HTML input section.

6 This part is used to write an SQL query. You can use standard SQL query statements. DB2 WWW Connection V1 supports SELECT, INSERT, DELETE, and UPDATE SQL statements.

7 This part is used to customize how the query results will look on the screen in HTML format. In this case, we are using tables to list the selected records. N1, N2, ... N5 are variables which are set by the system. The value of these variables are the names of columns selected in the SQL query. N*n* is valid only inside the SQL Report section.

8 Part of SQL Report subsection is the Row subsection. In this subsection, we define how each row returned by the SQL query will be displayed. We use V*n* variables which are also variables set by the system. The value of V1, V2, ... V5 will change each time a new row is retrieved. These variables are valid only inside a Row subsection.

6, **7** and **8** are the SQL Section, while SQL_REPORT is a subsection. If you don′t have an SQL_REPORT subsection, selected records will be shown in a default table with column names at the top.

You can also have an SQL_MESSAGE subsection that allows the customization of error and warning messages in SQL statements.

If you have more than one SQL section in a macro file, name them like this:

```
%SQL(Statement1){.......
%SQL(Statement2){.......
```

9 The HTML Report section is where the query is executed. If you have more than one SQL Section in your macro file, specify their names in the EXEC_SQL line:

```
%EXEC_SQL(Statement1)
%EXEC_SQL(Statement2)
```

You can define how your WWW screen looks in the HTML Report Section by using standard HTML. For example, you can give a new title, new header, some comments and references, or icons that are different from the input page (defined in HTML Input section).

Not all the sections mentioned in 3.4.1, “Macro Files” on page 24 are used. Some of them are optional, and if you use them or not depends on your application design. Also, the order in which you write each section doesn′t need

to be the same as this sample. You can write them in any order. Logically, the HTML Input section will be read first. After variables are submitted by a user, the application calls the URL written in the ACTION field, as shown in **2**, which is usually the HTML Report section. The HTML Report section then executes the SQL query and shows the result on its defined form.

Figure 31 is the output screen from running the above macro file. The input screen is the same as the CGI script sample which can be seen in Figure 28 on page 31.

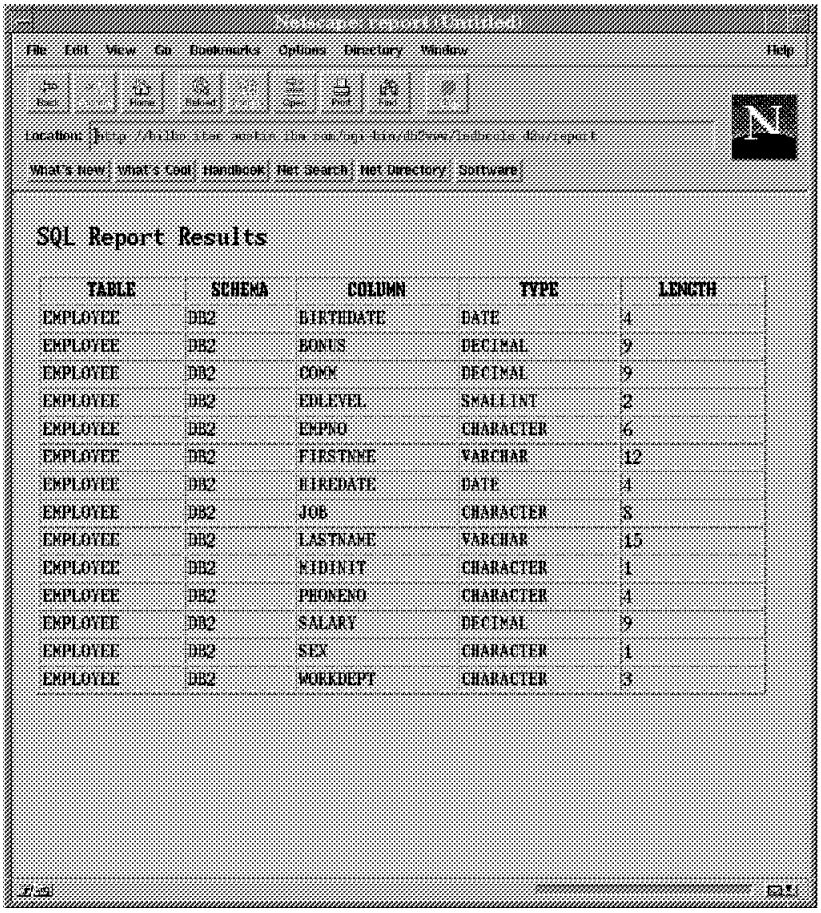


TABLE	SCHEMA	COLUMN	TYPE	LENGTH
EMPLOYEE	DB2	BIRTHDATE	DATE	4
EMPLOYEE	DB2	BONUS	DECIMAL	9
EMPLOYEE	DB2	COMM	DECIMAL	9
EMPLOYEE	DB2	EDLEVEL	SMALLINT	2
EMPLOYEE	DB2	EMPNO	CHARACTER	6
EMPLOYEE	DB2	FIRSTNME	VARCHAR	12
EMPLOYEE	DB2	HIREDATE	DATE	4
EMPLOYEE	DB2	JOB	CHARACTER	8
EMPLOYEE	DB2	LASTNAME	VARCHAR	15
EMPLOYEE	DB2	MIDINIT	CHARACTER	1
EMPLOYEE	DB2	PHONENO	CHARACTER	4
EMPLOYEE	DB2	SALARY	DECIMAL	9
EMPLOYEE	DB2	SEX	CHARACTER	1
EMPLOYEE	DB2	WORKDEPT	CHARACTER	3

Figure 31. Output Screen from Macro File Sample

3.6.3 Using LOBs in a Macro File

The following sample shows you how to work with LOBs stored in DB2 databases and how to present them in a WWW browser application.

```
%define DATABASE="sample"
%{qemp.d2w Query database to find out employee information.
%}
%define{te="employee"
         tq="db2"
         tp="emp_photo"
         tr="emp_resume"
         docroot="/usr/local/www/pub"                          1
         move=%exec "mv $(docroot)$(V5) $(docroot)$(V5).txt"
%}

%SQL{
select e.empno, e.firstnme, e.lastname, p.picture,
       r.resume from $(tq).$(te) e, $(tq).$(tp) p, $(tq).$(tr) r
where  e.empno=p.empno and e.empno=r.empno and
       e.firstnme =ucase('$(InputName)') and
       p.photo_format = 'gif' and r.resume_format = 'ascii'  2

%SQL_REPORT{
%ROW{
$(move)
<FORM METHOD="POST" ACTION="/cgi-bin/db2www/qemp.d2w/report">
<PRE>
Employee No. : $(V1)
  First Name : $(V2)
   Last Name : $(V3)
       Photo :
<P>            <IMG SRC="$(V4)">                    3
<P>   Resume : <a href="$(V5).txt">Click here to read $(V3)'s resume</A>
<br> </PRE>
<INPUT TYPE="submit" VALUE="UPDATE RECORDS">
<INPUT TYPE="reset"  VALUE="RESET VALUES">
</FORM>
%}
%}

%SQL_MESSAGE{
100 :"<strong>WARNING</strong>:
No employees were found that met your search criteria.<p>": continue
%}
%}

%HTML_INPUT{
<TITLE>DB2 WWW Customer Information Query</TITLE>
<FORM METHOD="POST"  ACTION="/cgi-bin/db2www/qemp.d2w/report">
<CENTER><H1>EMPLOYEE QUERY FORM</H1></CENTER>
<hr> <PRE>
Employee Name : <INPUT TYPE="text" NAME="InputName" SIZE=30 VALUE="DOLORES">
</PRE> <HR>
<INPUT TYPE="submit" VALUE="QUERY"> <INPUT TYPE="reset" VALUE="RESET">
</FORM>
<P>
%}

%HTML_REPORT{
<TITLE>DB2 WWW Information Results</TITLE>
<P><H1>EMPLOYEE DATA</H1><P>

%EXEC_SQL
<P> <HR>
%}
```

Figure 32. Using LOBs from DB2 Databases in a Macro File: qemp.d2w

1 First, you have to define the directory where the temporary file for LOBs will be saved `docroot="/usr/local/www/pub"`. Then rename the file with the correct extension according to its type `move=%exec "mv $(docroot)$(V5) $(docroot)$(V5).txt"`

2 Now, build the query statement matching the column with the variable name defined before (the select for the CLOB field must be in the fifth position in the query columns).

3 In the report form, you can use a specific viewer for each data type, `<IMG SRC="$(V4)">` in case of pictures, or let the browser define which kind of data it will present based on the file extension, `<a href="$(V5).txt">`.

Everything else works as it was explained in 3.6.2, "DB2 WWW Macro Example" on page 32.

The following figures show the result of running the above macro file. The first screen is Figure 33. And if you click on the sentence **Click here to read QUINTANA's resume**, you will have the resume text as seen in Figure 34 on page 38.

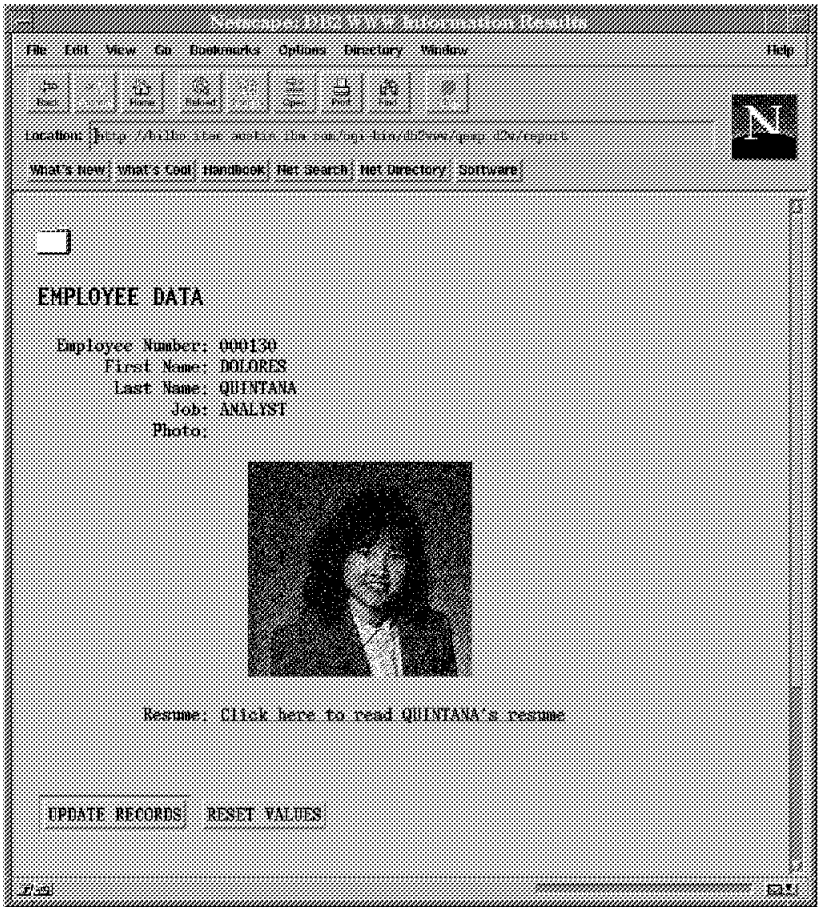


Figure 33. First Screen Resulting from the qemp.d2w Macro File

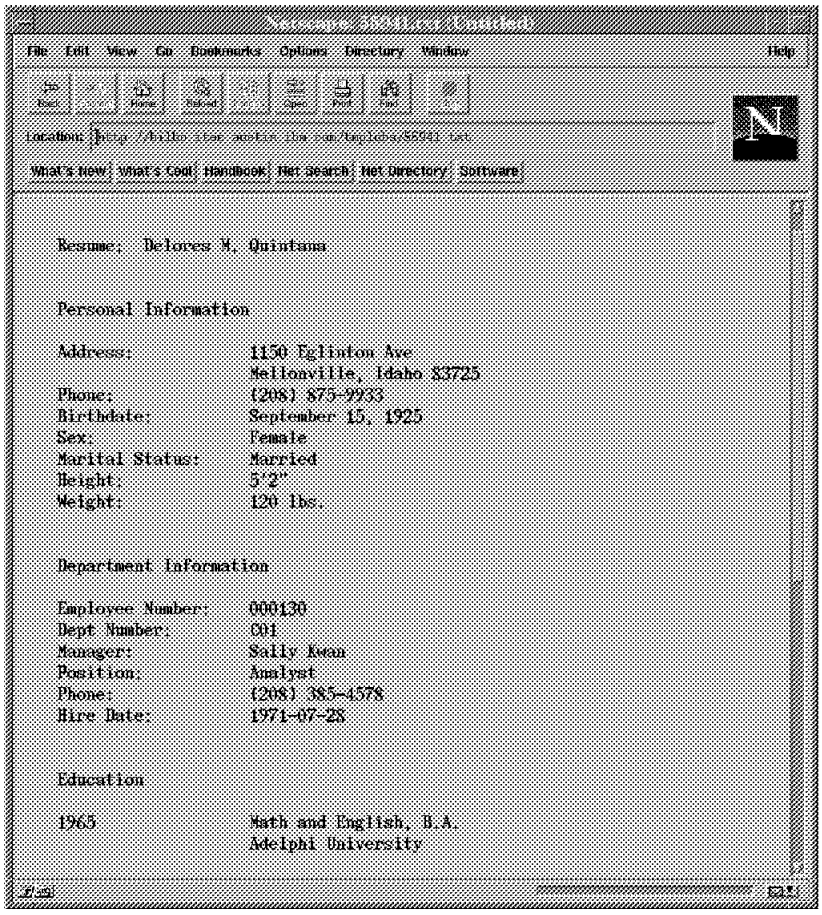


Figure 34. Second Screen Resulting from the qemp.d2w Macro File Shows the Resume

Chapter 4. Call Level Interface

There are several ways to write applications to access DB2 databases. This chapter discusses writing applications using the DB2 Call Level Interface (DB2 CLI), which is provided in DB2 Version 2. The discussion includes information about DB2 CLI, how to write the application, what should be considered when writing applications using CLI, and some examples. The full listing of the code fragments found in this chapter have been included in Appendix A, “Sample Applications” on page 143.

4.1 Overview

DB2 Call Level Interface (DB2 CLI) is a callable SQL interface for the DB2 family of database servers. It provides an alternative way to access DB2 Common Server databases by using SQL statements through function calls instead of using embedded SQL in the application. The advantage in using DB2 CLI is that you do not have to precompile the application, as is required by embedded SQL. Your application will be more portable since it does not depend on any particular precompiler. This is because DB2 CLI applications use a common access package provided with any DB2 for Common Server product. These differences are discussed in further in 4.1.1, “ Differences between DB2 CLI and Embedded SQL.”

To access a database, you code function calls with the DB2 CLI to invoke dynamic SQL statements. DB2 CLI will pass the SQL statements to the database manager for processing. When coding your application, you need to consider the data types used in the transactions, the results and deal with error handling. These areas are covered throughout this chapter.

4.1.1 Differences between DB2 CLI and Embedded SQL

DB2 CLI and Embedded SQL applications are different in the following ways:

- A DB2 CLI application does not need to be precompiled or bound. It uses a standard set of functions to execute SQL statements at run time.

  This is the most important difference because this makes the DB2 CLI application independent of any particular database product. You do not need to recompile or bind your DB2 CLI application to access different DB2 databases.

- DB2 CLI generates required cursors automatically, instead of having to explicitly declare the cursor as required in Embedded SQL applications.

- DB2 CLI does not use the open statement. Executing a SELECT statement will automatically open a cursor.

- DB2 CLI allows the use of parameter markers in the SQLExecDirect() function. This is the equivalent of the EXECUTE IMMEDIATE statement in Embedded SQL.

- DB2 CLI uses the SQLTransact() function call to issue a COMMIT or ROLLBACK statement.

- DB2 CLI manages environment handles, connection handles and statement handles. Detailed information about these handles is discussed in 4.2, “ Writing DB2 CLI Applications” on page 41.

- DB2 CLI uses SQLSTATE values defined by the X/Open SQL common applications environment specification. This ensures the application will get consistent message handling across different database servers.

DB2 CLI is suited for creating applications in a client/server environment in which the target database is not known when the application is built. It is also well suited for applications that require portability, since they do not need to be precompiled.

DB2 CLI also supports multiple connections to multiple databases or multiple connections to the same database, since each connection may have its own commit scope. This can be achieved because DB2 CLI does not need to use application controls, such as SQLDA and SQLCA, which are typically associated with Embedded SQL applications. DB2 CLI allocates and controls the necessary data structures and provides a handle for referencing them. This enables DB2 CLI to create multi-threaded applications where each thread can have its own connection and a separate commit scope to any other connections or threads.

One consideration when using the DB2 CLI interface is that DB2 CLI applications do not use static SQL. Static SQL is a statement that is fully known at precompile time. In contrast to static SQL, a dynamic SQL statement is not fully known until run time. Only embedded applications will use static SQL.

You may decide to use static SQL in your application because it has the following characteristics:

- Static SQL may have better performance than dynamic SQL because dynamic SQL needs more processing time, and the preparation step of dynamic SQL may cause additional network-traffic at run time.
- In using static SQL, users do not need to have access to the data objects (such as tables, views, columns, and so on). Instead, the authorization of the objects are associated with a package and validated at the package binding time. The privileges of the person who performs the binding of the application will be used when other users execute it. Other users only need the execute privilege for the static package.

You might already know the SQL statement and use the SQLExecute() function to pass the complete SQL statement to the database in a DB2 CLI application. This will not provide you with any of the features of static SQL because the statement is still dynamic since access plans and optimization will still need to be performed at run time.

In some cases, you may need both the advantages of using static SQL and DB2 CLI. This can be achieved because DB2 CLI applications can call stored procedures on the server that may contain static SQL. How to use stored procedures in a DB2 CLI application is discussed in 4.4.7, “Stored Procedures” on page 84.

4.1.2 Supported Environments

To create and use DB2 CLI applications, we need both of the following:

1. The development tools

   DB2 CLI development support is included with DB2 Software Developer′s Kit (DB2 SDK) products, which consists of the necessary header files, link libraries, and documentation required to develop both Embedded SQL and

DB2 CLI applications. You can develop the application using DB2 SDK installed on a DB2 server or on a DB2 client workstation.

2. The runtime

   DB2 server products and DB2 Client Application Enabler (DB2 CAE) products support the runtime for DB2 CLI application. The CAE is included as a component of the SDK product.

Figure 35 shows an example of an OS/2 environment that uses DB2 CLI applications. Developers use DB2 SDK for OS/2 to write DB2 CLI applications that run under OS/2. Users can use either DB2 CAE for OS/2 or DB2 for OS/2 to run the application. This application can access data on DB2 for OS/2, DB2 for AIX or any other DB2 for Common Server. And if you use Distributed Database Connection Services (DDCS), the application can also access DB2 for OS/400, DB2 for MVS/ESA, DB2 for VSE and VM servers, or any Distributed Relational Database Architecture (DRDA) servers.

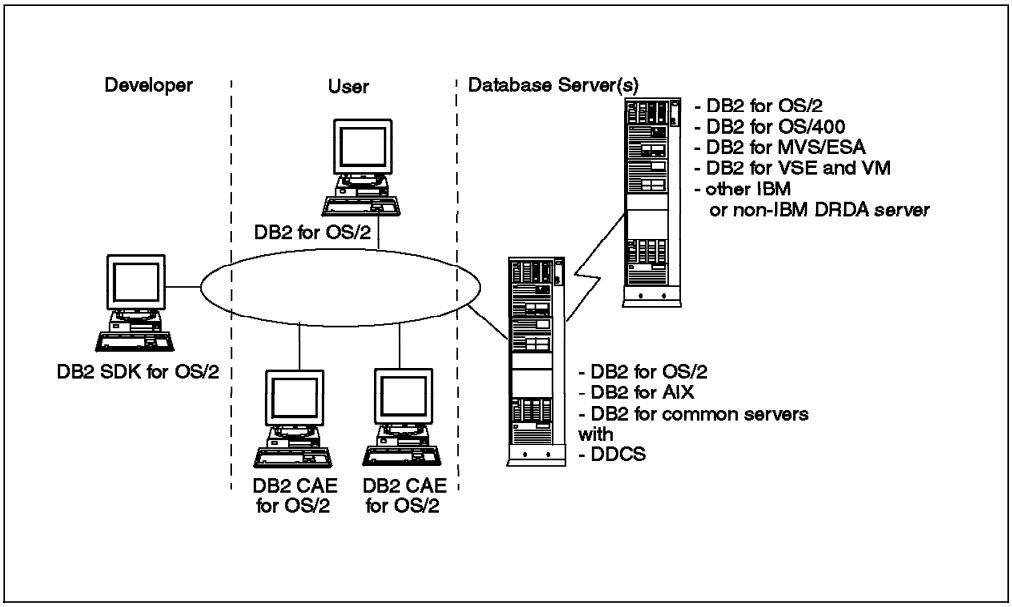


Figure 35. Example of a DB2 CLI Environment

4.2 Writing DB2 CLI Applications

Basically, a DB2 CLI application consists of sets of tasks. Each task is carried out by calling one or more DB2 CLI functions.

In this section, we discuss the basic tasks that apply to all DB2 CLI applications. There are also general tasks, such as handling diagnostic messages, which occur throughout an application.

For further reference on the topics covered in this chapter, you should refer to *Call Level Interface Guide and Reference - for common servers*.

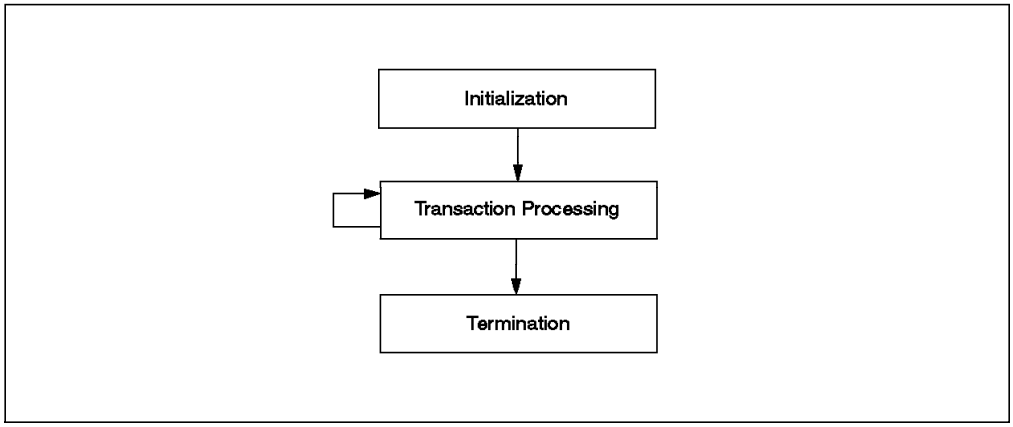


Figure 36. Basic Tasks in a DB2 CLI Application

As shown in Figure 36, the basic tasks in a DB2 CLI application are:

1. Initialization
2. Transaction Processing
3. Termination

Before discussing each task, there is an important term that is used in DB2 CLI, and that is the term “handle.”

A *handle* is a variable that refers to a data object controlled by DB2 CLI. By using handles, DB2 CLI applications do not have to allocate and manage global variables or data structures, such as the SQLDA or SQLCA, as used in the Embedded SQL interfaces.

There are three types of handles:

- **Environment Handle** refers to the data object that contains information regarding the global state of the application, such as attributes and connections.
- **Connection Handle** refers to a data object that contains information associated with a connection to a particular database, such as connection options, general status information, transaction status, and diagnostic information. An application requires a connection handle for each concurrent connection to a database server.
- **Statement Handle** refers to the data object that contains information associated with the execution of a single SQL statement, such as statement options, dynamic parameters, cursor information, bindings for dynamic arguments and columns, result values, and status information. Each statement handle is associated with a connection handle.

4.2.1 Initialization and Termination

The initialization task allocates and initializes the environment and connection handles, while the termination task will free the allocated handles.

An environment handle must be allocated before a connection can be allocated. To allocate an environment handle, we use `SQLAllocEnv()`. To free it, we use `SQLFreeEnv()`.

A connection handle is allocated by calling SQLAllocConnect() and freed by calling SQLFreeConnect().

You have to allocate more than one connection handle in an application if the application uses concurrent connections to a single database. This is because each connection needs its own connection handle.

Figure 37 shows a conceptual view of Initialization and Termination tasks.

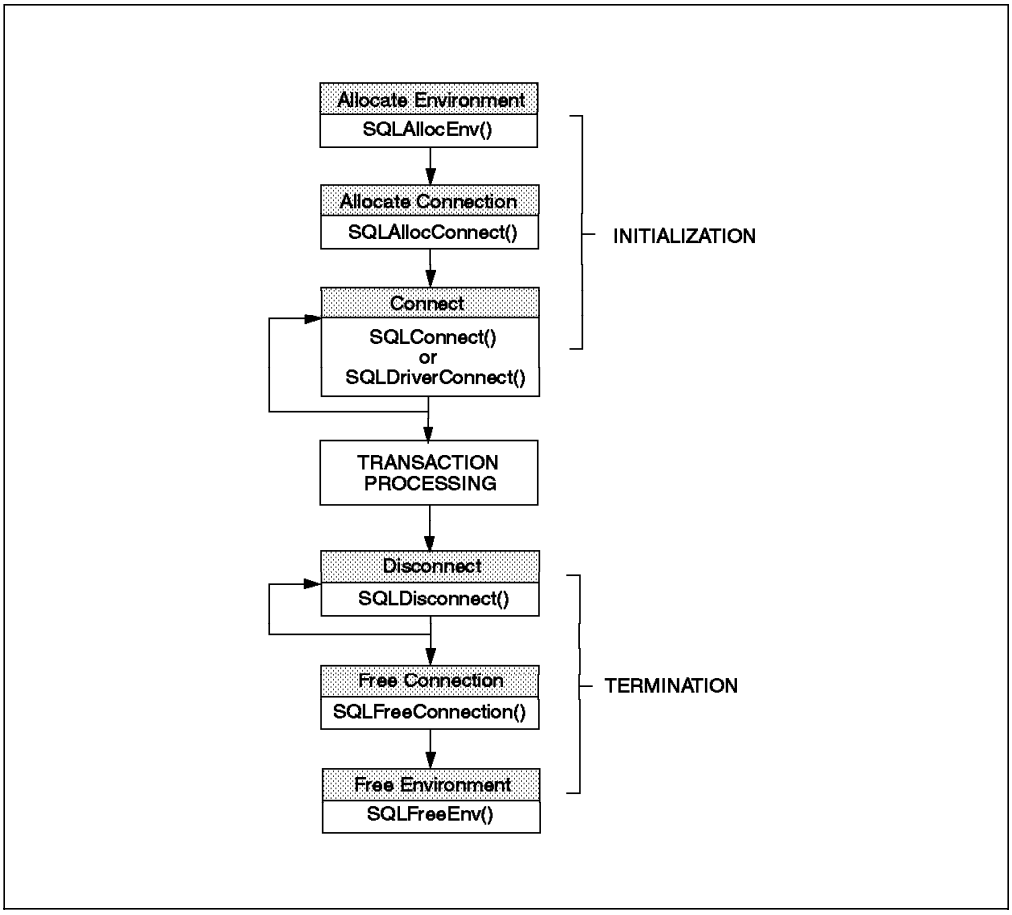


Figure 37. Conceptual View of Initialization and Termination tasks

4.2.2 Transaction Processing

Transaction Processing is the main task of a DB2 CLI application. It passes SQL statements to DB2 CLI in order to query or modify the data.

Transaction Processing consists of the following five steps:

1. Allocating Statement Handle(s)
2. Preparation and Execution of SQL Statements
3. Processing Results
4. Commit or Rollback
5. Freeing Statement Handle(s)

Figure 38 on page 44 shows each function associated with the above steps.

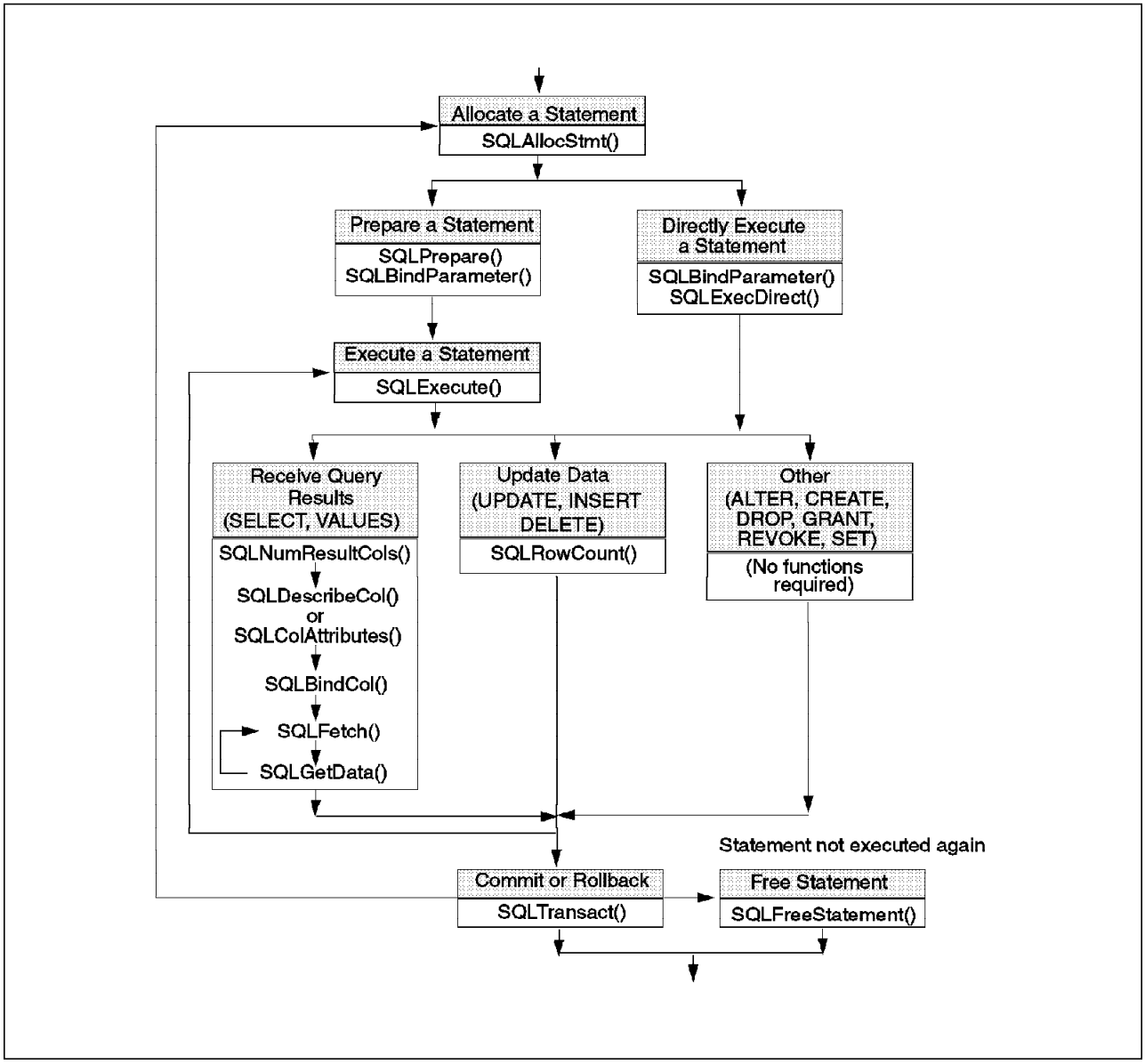


Figure 38. Transaction Processing Task

Allocating Statement Handle(s)

Before executing an SQL statement, an application needs to allocate a statement handle. A statement handle is allocated by calling the `SQLAllocStmt()` function.

Preparation and Execution

There are two available methods that can be used in this step:

1. Prepare then Execute:

   In this method, the preparation of a statement is split from the execution. We use this method when the statement is going to be executed repeatedly, and the application needs information about the columns in the result set (a set of row(s) and/or column(s) resulting from a query) before the statement is executed. Using this method, we do not have to prepare the same statement more than once.

2. Execute Direct:

   In this method, we combine the preparation and execution steps into a single step. We use this method when the statement is going to be executed only once, and the application does not need the information about the columns in the result set before the statement is executed. Using this method, we do not need to call two functions to execute the statement.

The functions used in each method are shown in the Figure 38 on page 44.

Either the Prepare then Execute or Execute Direct method support the use of parameter markers in an SQL statement. A *Parameter marker* is a marker used to indicate the position of an application variable that will be used in an SQL statement when the statement is executed. This function is similar to the use of host variables in an Embedded SQL application. The application must bind an application variable to each parameter marker used in the SQL statement by calling the SQLBindParameter() or SQLSetParam() functions. The parameter markers are indicated by "?" characters in the SQL statement and are referenced sequentially from left to right. For example, if you define an SQL statement with two parameter markers:

```
SELECT tabname, tabschema, colname, typename, length \
       from syscat.columns where tabname = ? and tabschema = ?
```

Then you need to call the SQLBindParameter() function for each parameter marker to be bound, as shown below:

```
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 10,
                 0, table_name.s, 10, SQL_NULL_DATA );

SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 10,
                 0, table_schem.s, 10, SQL_NULL_DATA );
```

Processing Results

An SQL statement in a DB2 CLI application may be one of the following types:

1. Processing Query Statement (SELECT, VALUES)

   To retrieve rows resulting from a query statement, a DB2 CLI application generally uses the following steps:

   a. Describe the structure of the result set, number of columns, column types, and lengths

      If the statement is generated by the application, it might not use this step since the application may know the structure of the result set and the data types of each column.

      But if the statement is generated at run time, the application needs to know the information about the number of columns, the names and data types of each column in the result set. This information can be obtained by calling SQLNumResultCols(), SQLDescribeCol() SQLColAttributes() after preparing or executing the statement.

   b. Bind application variables to columns to receive the data

      To bind an application variable to a column in the result set, we use SQLBindCol(). This allows the application to retrieve data on a column directly into an application variable on the next call of SQLFetch(). You need to call SQLBindCol() for each column to be retrieved. You can also

use this function to determine the C data type to be used in the application variable.

This step is optional since an application does not bind any columns when it needs to retrieve columns of large amounts of data in pieces. In this case, the application uses SQLGetData().

c. Repeatedly fetch the next row of data and receive it into the bound variables

In this step, the application calls SQLFetch() to fetch the first or next row of the result set. If the application has determined the C data type to be used in the application variable in the SQLBindCol(), data conversion from SQL data types to C data types occurs here.

d. Retrieve columns that were not previously bound

This step is also an optional step. This step is used when the application does not bind columns. The function used is SQLGetData(), and the application needs to call this function for each fetch by using SQLFetch(). As in SQLBindCol(), you can also specify the C data type to be used for the resulted data.

2. Processing UPDATE, INSERT and DELETE Statements

There is no action needed in processing the results of modifying data statements, such as UPDATE, INSERT and DELETE, other than the normal check for diagnostic messages. You do not have to call any functions to get the structure of a result set, bind application variables to columns or fetch data. This is because there is no data returned from these statements. You may only need to call only the SQLRowCount() function which is used for obtaining the number of rows affected by the SQL statement.

If the statement is a positioned UPDATE or DELETE, it requires the use of a cursor, that is, a pointer to a row in the result table of an active query statement. In DB2 CLI applications, the cursor name is automatically generated when calling SQLAllocStmt(). To use this cursor in a positioned UPDATE or DELETE, we call the SQLGetCursorName() function.

3. Processing Other Statements

Other statements that are not queries or modifying data statements, such as CREATE, ALTER, GRANT, and REVOKE, do not need further action other than the normal check for diagnostic messages.

Commit or Rollback

DB2 CLI supports two types of commit mode: *auto-commit* and *manual-commit*. The default commit mode is auto-commit, but we can switch between auto-commit and manual-commit in an application by calling the SQLSetConnect() function.

Table 6 on page 47 summarizes the difference between auto-commit and manual-commit.

Table 6. The Difference between Auto-Commit and Manual-Commit	
Auto-Commit	**Manual-Commit**
A transaction is an SQL statement which will be automatically committed at the end of the statement execution	A transaction is started with the first access to a database using SQLPrepare(), SQLExecDirect(), or any functions that returns a result set and ends using SQLTransact() to commit or rollback the transaction
Does not need to call SQLTransact()	Must call SQLTransact() to commit or rollback a transaction
Typically used in a query-only application	Typically used in an application that performs updates

Freeing Statement Handle(s)

This step is used to end processing for a particular statement handle. The function called is SQLFreeStmt(). We call this function when the statement is not going to be executed again. It will do one or more of the following:

- Unbind all columns of the result set
- Unbind all parameter markers
- Close any cursors and discard any pending results
- Drop the statement handle and release all associated resources

4.2.3 Diagnostics and Error Handling

Diagnostic information is used to handle warning and error conditions generated within a DB2 CLI application by calling a CLI function. DB2 CLI provides two levels of diagnostic:

1. Return Codes

   The execution of a CLI function causes one or more conditions to be raised. The basic result of the execution is indicated by a code that is returned by the called function.

   Table 7 below lists all possible return codes that might be generated by a DB2 CLI function.

Table 7 (Page 1 of 2). Possible Return Codes from a DB2 CLI Function	
Return Code	**Description**
SQL_SUCCESS	The function completed successfully; no additional SQLSTATE information is available.
SQL_SUCCES_WITH_INFO	The function completed successfully with a warning or other information. Use SQLError() to get the other information.
SQL_NO_DATA_FOUND	The function returned successfully, but no relevant data was found. Call SQLError() to obtain additional information.
SQL_NEED_DATA	The application tried to execute an SQL statement, but DB2 CLI lacks parameter data that the application had indicated would be passed at execute time.

Table 7 (Page 2 of 2). Possible Return Codes from a DB2 CLI Function	
Return Code	**Description**
SQL_ERROR	The function failed. Use SQLError() to obtain SQLSTATE and other information.
SQL_INVALID_HANDLE	The function failed due to an invalid input handle (environment, connection or statement handle) caused by a programming error. No further information is available.

2. Detailed Diagnostics

   Detailed diagnostics can be accessed by calling one of the following:

   - SQLError() which provides diagnostic information associated with the most recently invoked DB2 CLI function for a particular statement, connection or environment handle. The information returned by SQLError() consists of a standardized SQLSTATE, native error code and a text message.
   - SQLGetSQLCA() which provides SQLCA associated with the preparation and execution of an SQL statement, fetching data or closing a cursor.

DB2 CLI provides a standard set of SQLSTATE values defined by the X/Open SQL CAE specification. This means that the application will receive consistent message handling across different database servers (usually different database servers have different diagnostic message codes).

Figure 39 shows the use of Return Code after calling an SQLConnect() function. And when it does not return SQL_SUCCESS, it will call SQLError() function to get additional information about the error.

```
rc = SQLConnect(hdbc, dbname, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
if (rc != SQL_SUCCESS) {
    printf("--- ERROR while connecting to database : %s ---\n", dbname);
    while (SQLError(henv, hdbc, hstmt, sqlstate, &sqlcode, buffer,
                    SQL_MAX_MESSAGE_LENGTH + 1, &elength) == SQL_SUCCESS){
           printf("         SQLSTATE: %s\n", sqlstate);
           printf("Native Error Code: %ld\n", sqlcode);
           printf("%s \n", buffer);
    };
    return (SQL_ERROR);
    } else {
           printf("\n--- Connected to database : %s ---\n\n", dbname);
    };
```

Figure 39. Return Code and Detailed Diagnostic Sample

Figure 40 on page 49 shows the result of calling SQLError() when using the sample code listed in Figure 39.

```
Enter Database Name : sampler
Enter User Name     : db2
Enter Password for db2 : db2
--- ERROR while connecting to database : sampler ---
         SQLSTATE: 08001
Native Error Code: -1013
[IBM][CLI Driver] SQL1013N  The database alias name or database name
"SAMPLER" could not be found.  SQLSTATE=42705
```

Figure 40. Information Returned from Calling SQLError() Function

4.2.4 Data Types and Data Conversion

In writing DB2 CLI applications, we have to deal with both SQL data types used by the database management service (DBMS) and C data types used in the application. When calling DB2 CLI functions, the application must match C data types to SQL data types when transferring data between the DBMS and the application.

To meet the above requirement, DB2 CLI:

- Provides symbolic names for the various data types
- Manages the transfer of data between the DBMS and the application
- Converts data when required (for example, converts a C character string to an SQL INTEGER type)

Before DB2 CLI performs the data transfer between the DBMS and the application, we have to identify the source, the target, or both data types by calling the `SQLBindParameter()`, `SQLBindCol()` or `SQLGetData()` functions. These functions use the symbolic type names. Table 8 on page 50 lists the SQL data types in :orrespondence with its Symbolic Data Types and Default C Symbolic Data Types.

An explanation for each column in the table is as follows:

1. The **SQL Data Type** column contains the SQL data type as it appears in the SQL DDL CREATE statement. The SQL data type is dependent in the DBMS.
2. The **Symbolic SQL Data Type** column contains a SQL symbolic name. The Symbolic SQL Data Type is defined as an integer value which is used by various functions to identify the SQL data types listed in the SQL Data Type column.
3. The **Default C Symbolic Data Type** column contains the C symbolic name, which is also defined as an integer value. The symbolic names are used by various functions to indicate the C data types of the application variables. We can indicate the C data types by explicitly specifying them in the arguments of a function or by specifying SQL_C_DEFAULT in the arguments, which will tell DB2 CLI to take the default C data types based on the SQL data type of column. For example, the default C data type of SQL_DECIMAL is SQL_C_CHAR.

Table 8. SQL Symbolic and Default Data Types

SQL Data Type	Symbolic SQL Data Type	Default Symbolic C Data Type
BLOB	SQL_BLOB	SQL_C_BINARY
BLOB LOCATOR	SQL_BLOB_LOCATOR	SQL_C_BLOB_LOCATOR
CHAR	SQL_CHAR	SQL_C_CHAR
CHAR FOR BIT DATA	SQL_BINARY	SQL_C_BINARY
CLOB	SQL_CLOB	SQL_C_CHAR
CLOB LOCATOR	SQL_CLOB_LOCATOR	SQL_C_CLOB_LOCATOR
DATE	SQL_DATE	SQL_C_DATE
DBCLOB	SQL_DBCLOB	SQL_C_DBCHAR
DBCLOB LOCATOR	SQL_DBCLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR
DECIMAL	SQL_DECIMAL	SQL_C_CHAR
DOUBLE	SQL_DOUBLE	SQL_C_DOUBLE
FLOAT	SQL_FLOAT	SQL_C_DOUBLE
GRAPHIC	SQL_GRAPHIC	SQL_C_DBCHAR
INTEGER	SQL_INTEGER	SQL_C_LONG
LONG VARCHAR	SQL_LONGVARCHAR	SQL_C_CHAR
LONG VARCHAR FOR BIT DATA	SQL_LONGVARBINARY	SQL_C_BINARY
LONG VARGRAPHIC	SQL_LONGVARGRAPHIC	SQL_C_DBCHAR
NUMERIC	SQL_NUMERIC	SQL_C_CHAR
REAL	SQL_REAL	SQL_C_FLOAT
SMALLINT	SQL_SMALLINT	SQL_C_SHORT
TIME	SQL_TIME	SQL_C_TIME
TIMESTAMP	SQL_TIMESTAMP	SQL_C_TIMESTAMP
VARCHAR	SQL_VARCHAR	SQL_C_CHAR
VARCHAR FOR BIT DATA	SQL_VARBINARY	SQL_C_BINARY
VARGRAPHIC	SQL_VARGRAPHIC	SQL_C_DBCHAR

As mentioned previously, DB2 CLI can also perform data conversion. Table 8 shows only the default data conversions. The SQLBindParameter(), SQLBindCol() and SQLGetData() functions can be used to convert data not only to the default data type but also to other data types. Table 9 on page 51 lists all the data type conversions supported by DB2 CLI.

Table 9. Supported Data Conversions

SQL Data Type	SQL_C_CHAR	SQL_C_LONG	SQL_C_SHORT	SQL_C_TINYINT	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_DATE	SQL_C_TIME	SQL_C_TIMESTAMP	SQL_C_BINARY	SQL_C_BIT	SQL_C_DBCHAR	SQL_C_CLOB_LOCATOR	SQL_C_BLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR
BLOB	X									D				X	
CHAR	D	X	X	X	X	X	X	X	X	X	X				
CLOB	D									X			X		
DATE	X						D		X						
DBCLOB										X		D			X
DECIMAL	D	X	X	X	X	X					X				
DOUBLE	X	X	X	X	X	D					X				
FLOAT	X	X	X	X	X	D					X				
GRAPHIC	X											D			
INTEGER	X	D	X	X	X	X					X				
LONG VARCHAR	D									X					
LONG VARGRAPHIC	X									X		D			
NUMERIC	D	X	X	X	X	X					X				
REAL	X	X	X	X	D	X					X				
SMALLINT	X	X	D	X	X	X					X				
TIME	X							D	X						
TIMESTAMP	X						X	X	D						
VARCHAR	D	X	X	X	X	X	X	X	X	X	X				
VARGRAPHIC	X											D			

Note:

D The default data type conversion.

X Other data types conversion.

blank Not a supported data type conversion.

For example, by default, the BLOB SQL data type is converted to SQL_C_BINARY. But BLOB SQL data type can also be converted to SQL_C_CHAR or SQL_C_BLOB_LOCATOR if you define it. These three data types are the only C types that the BLOB SQL data type can be converted to.

DB2 CLI supports the use of User-Defined Types (UDTs) which is provided in DB2 Version 2. Data conversion involving UDTs in a DB2 application is covered in 4.4.6, “User-Defined Types (UDTs)” on page 83.

4.3 CLI Application Configuration and Execution

Support for DB2 Call Level Interface applications is provided in all of the DB2 for Common Server Products. The *Client Application Enabler* provides the runtime environment, while with the *Software Developer's Kit*, you can also perform application development and testing.

4.3.1 Setting the DB2 CLI Runtime Environment

To successfully access a DB2 database from any DB2 CLI application, you need to consider the following:

1. The database (and its node if the database is remote) must be cataloged. Use the Command Line Processor or DB2 administration tool, if applicable.

   For an example, using the CLP interface, assume the TCP/IP protocol is being used to connect to the remote server:

   ```
   db2=> CATALOG TCPIP NODE tcpnode REMOTE tcpserv SERVER tcpsvce
   ```

   To verify if it was successful, you can issue:

   ```
   db2=> LIST NODE DIRECTORY
   ```

   ```
   Node Directory

   Number of entries in the directory = 1

   Node 1 entry:

   Node name                      = tcpnode
   Comment                        =
   Protocol                       = TCP/IP
   Hostname                       = tcpserv
   Service name                   = tcpsvce
   ```

   To catalog the database after cataloging its remote node:

   ```
   db2=> CATALOG DATABASE tcpdb AS tcpdb AT NODE tcpnode
   ```

   To verify:

   ```
   db2=> LIST DATABASE DIRECTORY
   ```

   ```
   Local Database Directory

   Number of entries in the directory = 1

   Database 1 entry:

   Database alias                 = tcpdb
   Database name                  = tcpdb
   Local database directory       =
   Database release level         = 6.00
   Comment                        =
   Directory entry type           = Remote
   ```

   To test the connection from the CLP, you can use the command:

   ```
   db2=> connect to tcpdb
   ```

2. The DB2 CLI bind files must be bound to the database. The bind files needed by each server are listed in Table 10 on page 53.

Table 10. DB2 Call Level Interface Bind Files

Bind File Name	Package Name	Needed by DB2 Common Server	Needed by DRDA Servers
db2clics.bnd	SQLL15xx	Yes	Yes
db2clirr.bnd	SQLL25xx	Yes	Yes
db2cliur.bnd	SQLL35xx	Yes	Yes
db2clirs.bnd	SQLL45xx	Yes	Yes
db2clinc.bnd	SQLL55xx	No	DB2 for OS/400
db2cliws.bnd	SQLL65xx	Yes	No
db2clims.bnd	SQLL75xx	No	DB2 for MVS/ESA
db2clivm.bnd	SQLL85xx	No	SQL/DS
db2cliv1.bnd	SQLLB5xx	Version 1 only	No
db2cliv2.bnd	SQLL95xx	Version 2.1 only	No
db2clias.bnd	SQLLA5xx	No	DB2 for OS/400

Note: Where 'xx' is unique for each platform, such as:

C0 DB2 for AIX
D0 DB2 for OS/2
W0 DB2 Client Application Enabler for Windows

The db2cli.lst file contains the names of the required bind files for DB2 CLI to connect to DB2 Version 2 servers. The db2cliv1.lst file contains the bind list for DB2 Version 1 servers.

For DRDA servers, use one of ddcsvm.lst, ddcsmvs.lst, ddcsvse.lst, or ddcs400.lst bind list files.

To bind these files (using the CLP):

```
db2=> CONNECT TO tcpdb

db2=> BIND $path/@db2cli.lst
```

To bind .bnd files, use the following command:

```
db2=> BIND $path/db2clics.bnd
```

The `$path` is required only if `.lst` or `.bnd` files are not in sqllib/bnd directory.

DB2 CLI can also be configured using either the Database Director or The DB2 Client Setup administration tool, depending on your platform. Alternatively, you may edit the db2cli.ini file.

4.3.1.1 Configuring db2cli.ini

The db2cli.ini initialization file is an ASCII file which stores values for the DB2 CLI configuration options. Depending on the platform, it is stored in the following directory:

sqllib/cfg for UNIX

sqllib for OS/2

sqllib\win for Windows

The following is an example of a db2cli.ini file with two database alias sections:

```
; Database Alias Section 1.
[MYDB22]      1
AUTOCOMMIT=0                          2
TABLETYPE="'TABLE','SYSTEM TABLE'"

; Database Alias Section 2.           3
[MYDB2MVS]    1
DBNAME=SAAID
TABLETYPE="'TABLE'"
SCHEMALIST="'USER1',CURRENT SQLID,'USER2'"
```

1 is called the *section header* and is represented by the database alias written between squared brackets.

Each line, such as the one marked **2** (following the section header), is used to set the CLI parameters. Each keyword is associated with a keyword value, and these settings are applied only to the database alias named in the section header. The keywords are not case sensitive; however, their values may be if they are used as values to compare within a specific query.

When an application connects to a database that is not found in the db2cli.ini file, the default values are in effect. If you write duplicate entries for a keyword, the first will be used without any warning. The db2cli.ini options will be used unless the application overrides them.

You can introduce comment lines by placing a semicolon in the first position of a new line, as in **3**, and also have blank lines.

For most applications, it's not necessary to specify these keywords, but they can be used to:

- Help improve the performance or usability of an application
- Provide support for applications written for a previous version of DB2 CLI
- Provides specific work-arounds for existing ODBC applications

Table 11 shows the keywords that can be defined in the db2cli.ini file, their possible values and summarizes the way they affect the application behavior. For further information, refer to *Call Level Interface Guide and Reference*.

Table 11 (Page 1 of 5). DB2 Call Level Interface Configuration Keywords

Keyword	Value
AUTOCOMMIT	1 = on (default) 0 = off Default AUTOCOMMIT on means that each statement is treated as a complete transaction. You can set an alternative default, but this will only by used if the application does not specify another value as part of the program. You must be careful when overriding this default value because the application may depend on the default to operate properly.

Table 11 (Page 2 of 5). DB2 Call Level Interface Configuration Keywords

Keyword	Value
BITDATA	1 = report FOR BIT DATA and BLOB data types as binary data types (default) 0 = disabled Allows you to specify whether ODBC binary data types SQL_BINARY, SQL_VARBINARY, SQL_LONGVARBINARY and SQL_BLOB are reported as binary data types Only set BITDATA = 0 if you're sure that columns defined as FOR BIT DATA or BLOB contain character data, and the application cannot display binary data columns.
CONNECTTYPE	1 = multiple concurrent connections each with its own commit scope (default) 2 = coordinated connections with multiple databases in the same Distributed Unit Of Work (DUOW). It works with SYNCPOINT setting to determine if a Transaction Manager should be used. See also 4.4.1, " Distributed Unit of Work" on page 65.
CURRENTFUNCTIONPATH	*current_function_path* Defines the path used to resolve function and data type references used in dynamic SQL statements. The default is: "SYSIBM","SYSFUN","X" Where X is the value of the USER special register. The schema SYSIBM is always used first, unless specified elsewhere. The order of the schema names determines the order in which function names will be resolved.
CURRENTSQLID	*current_sqlid* It's valid for DB2 DBMS that support SET CURRENT SQLID (such as DB2 for MVS/ESA). This allows the end user and applications to name SQL objects without having to qualify by schema name.
CURSORHOLD	1 = Cursors are not destroyed when the transaction is committed (default). 0 = Cursors are destroyed after commit. Cursors are always destroyed when transactions are rolled back.
DB2ESTIMATE	0 = Estimates are not returned (default) *large positive number* = Is the threshold above which DB2 CLI will display the window to report estimates. If the value in SQLERRD(4) is greater than DB2ESTIMATE, the estimates window will appear. The recommended value is 60000.
DB2EXPLAIN	0 = Both off (default) 1 = Explain Snapshot Facility on 2 = Explain Table Information Capture Facility on 3 = Both on In each case, 'SET CURRENT EXPLAIN SNAPSHOT = YES/NO' and 'SET CURRENT EXPLAIN MODE = YES/NO' are sent to the server to enable or disable each facility. Explain tables must be created before the explain information can be generated and authorization ID must have INSERT privilege for these tables.
DB2OPTIMIZATION	*integer value from 0 to 9* Only applies to DB2 Version 2 server. If specified, DB2 CLI will issue the following statement after a successful connection: SET CURRENT QUERY OPTIMIZATION integer value This value represents the level at which the optimizer should operate the SQL queries. See *Adjusting the Optimization Class* in the *DB2 Administration Guide*.

Table 11 (Page 3 of 5). DB2 Call Level Interface Configuration Keywords	
Keyword	**Value**

DBALIAS	*dbalias*
	The maximum length for a database alias name is eight single byte characters. If you need to specify a longer alias name (to make it meaningful), it can be placed in the *section header* written between brackets, and then you must use this keyword to define the eight character alias name, as in the following example:
	`; Using a long database alias name`
	`[LongDatabaseName]`
	`DBALIAS=DB2DBXXX`

DBNAME	*dbname* Only used when connecting to DB2 for MVS/ESA and if (base) table catalog information is requested by the application. Can be specified to reduce the time it takes for the database to process the catalog query for table information and reduce the number if tables returned to the application. See also TABLETYPE.
GRAPHIC	0 = disabled (default) 1 = enabled 2 = Reports the length of graphic columns returned by SQLDescribe() in number of bytes rather than in Double-Byte Character Set (DBCS) characters. Applicable to DB2 CLI/ODBC functions that return length/precision in the output argument or as part of the result set. Needed for Microsoft Access 1.1 and Microsoft Query.
LOBMAXCOLUMNSIZE	*integer greater than 0* It will override the 2 GB (1 GB for DBCLOB) value returned by SQLGetTypeInfo() for the COLUMN_SIZE column for SQL_CLOB, SQL_BLOB and SQL_DBCLOB data types.
LONGDATACOMPAT	0 = no (default) References LOB data types as SQL_BLOB, SQL_CLOB and SQL_DBCLOB. 1 = yes References LOB data types as SQL_LONGVARCHAR, SQL_LONGVARBINARY and SQL_LONGVARGRAPHIC.
MAXCONN	0 | *positive number* Specifies the maximum number of connections allowed for each CLI application program. A value of 0 represents no limit; that means your application can open as many connections as permitted by the system resources.
MODE	SHARE (default) Does not prevent concurrent application processes from executing operations at the application server. EXCLUSIVE (Not permitted for DRDA connections) Prevents concurrent application processes from executing operations at the application server unless they have the same authorization ID as the user holding the exclusive lock. This value may be overridden by the application settings at connect time. See *Lock Table Statement* in the *SQL Reference*.

Table 11 (Page 4 of 5). DB2 Call Level Interface Configuration Keywords

Keyword	Value
MULTICONNECT	0 = false Each SQLConnect() request by the application will result in a physical database connection. 1 = true All connections for the application are mapped to a single physical connection. All the statements are executed in the same transaction; so a rollback will rollback all the statements on all the connections for the application.
OPTIMIZEFORNROWS	*integer* Will append the 'OPTIMIZE FOR n ROWS' clause to every SELECT statement, where *n* is an integer larger than 0. Default is not to append this clause.
PATCH1	{0 |1 |2 |4 |8 |16 |... } Used to specify a work-around for known problems with 16-bit Windows 3.1 ODBC applications. If you want the work-around to be additive, add the values together to form the keyword value. For example, if you need 1, 4 and 8 work-arounds added, you should write: PATCH1 = 13 Default is 0; that means without work-arounds.
PWD	*password* Defines the password to be used if it's not provided by the application at connect time.
SCHEMALIST	"'schema1', 'schema2', ..." A list of schemas in the database. It replaces the OWNERLIST keyword used in previous releases (still supported but not recommended). See also TABLETYPE.
SYNCPOINT	1 = One phase (default) Used to commit independently the work done by each database in a multiple database transaction. 2 = Two phase Specifies that a Transaction Manager is needed to coordinate two phase commits among those databases that support this. This keyword is relevant only if CONNECTTYPE = 2. See also 4.4.1, " Distributed Unit of Work" on page 65.
SYSSCHEMA	*sysschema* Indicates an alternative schema to be searched in place of the SYSIBM (or SYSTEM, QSYS2) schemas when DB2 CLI and ODBC catalog function calls are issued to obtain system catalog information. It replaces SYSOWNER keyword used in previous releases (still supported but not recommended).
TABLETYPE	"'TABLE'|, 'ALIAS'|, 'VIEW'|, 'INOPERATIVE VIEW'|, 'SYSTEM TABLE'|, 'SYNONYM'" Can be used with DBNAME and SCHEMALIST to limit the number of tables for which information will be returned.
TRANSLATEDLL	X:\PATH\DB2TRANS.DLL X:\PATH is the directory where DB2 CAE or SDK for Windows products have been installed. The DB2TRANS.DLL file contains codepage mapping tables.

Table 11 (Page 5 of 5). DB2 Call Level Interface Configuration Keywords	
Keyword	**Value**
TRANSLATEOPTION	*database codepage number* Lets TRANSLATEDLL and TRANSLATEOPTION enable the translation of characters from codepage number to the Windows 1004 codepage when working with DB2 Version 1 servers. Only two database codepage numbers are supported: 437 and 850. If you specify any different value, a warning will be returned during the connect request. This indicates that translation is not possible.
TNXISOLATION	1 = Read Uncommitted (Uncommitted read) 2 = Read Committed (Cursor Stability) (default) 4 = Repeatable Read (Read Stability) 8 = Serializable (Repeatable read) 32 = No Commit (DB2 for OS/400 only) Words in parenthesis are DB2 equivalents for SQL92 isolation levels. See *Application Processes, Concurrency and Recovery* in the *DB2 SQL Reference*.
UID	*user ID* Defines the user ID to be used if it's not provided by the application at connect time.
UNDERSCORE	1 = "_" acts as a wildcard matching any one character or none (default) 0 = "_" acts as itself. Setting this keyword to 0 will result in performance improvement in cases where object names in the database contain underscores. This keyword is valid only for versions prior to DB2 Version 2 server; this can use the ESCAPE clause for the LIKE predicate.

4.3.2 Application Development Environment Setup

DB2 CLI application development support is provided when you install DB2 SDK products. It requires the same initial runtime setup you defined at 4.3.1, "Setting the DB2 CLI Runtime Environment" on page 52.

To set up and verify the application development environment, do the following:

1. Verify the database (and its node if remote server is being used) are cataloged. Issuing the following commands will list the nodes and databases that have been cataloged.

   ```
   db2=> LIST NODE DIRECTORY
   db2=> LIST DATABASE DIRECTORY
   ```

2. Verify if connection is possible by issuing the following command:

   ```
   db2=> CONNECT TO tcpdb USER userId USING password
   ```

3. Verify the appropriate compiler is installed and set up.
4. Once the connection has been confirmed, you can compile DB2 CLI applications, as explained in 4.3.3, " Compiling and Linking Applications" on page 59.

   DB2 CLI includes some example applications in the /samples/cli directory under your DB2 SDK product installation directory.

5. After compilation is finished, without errors, you can execute the new application.

Section 4.3.4, " DB2 CLI Functions" on page 60 shows the functions used in DB2 CLI applications.

4.3.3 Compiling and Linking Applications

Since DB2 CLI is defined as a 'C' and 'C + + ' Application Programming Interface, applications can be written in any of those languages, and they also need to be compiled and linked.

There are some general considerations when programming DB2 CLI applications:

- The order in which the compiler searches for include (header) files can be significant if there are two or more files with the same name.
- If you're building only DB2 CLI applications, always put the DB2 include path before any other include files in your application.

If building ODBC applications, refer to Chapter 5, "Open Database Connectivity (ODBC)" on page 93.

Depending on which is your client operating system, you can choose to develop applications in IBM AIX XLC, IBM CSet++ for OS/2 or Microsoft Visual C.

Table 12, Table 13 and Table 14 show the differences that must be considered when you work in each application-development environment.

Table 12. Considerations When Programming in XLC for AIX

ini file	$DB2PATH/cfg/db2cli.ini
source file	$DB2PATH/samples/cli/clisampl.c
command file	$DB2PATH/samples/cli/clibld
compile	xlc -c clisampl.c -I/$DB2PATH/include
link	xlc -c clisampl clisampl.o /$DB2PATH/lib/libdb2.a -bloadmap:map

Note:

- DB2PATH is the directory where DB2 is installed.

Table 13. Considerations When Programming in CSet++ for OS/2

ini file	DB2PATH\db2cli.ini
source file	DB2PATH\samples\cli\clisampl.c
command file	DB2PATH\samples\cli\clibld.cmd
compile	icc -c clisampl.c
link	link386 /st:128000 /noi /clisampl.obj,clisampl.exe,NUL,db2cli;

Note:

- DB2PATH is the directory where DB2 is installed.

Table 14 (Page 1 of 2). Considerations When Programming in Microsoft Visual C for Windows

ini file	DB2PATH\win\db2cli.ini

Table 14 (Page 2 of 2). Considerations When Programming in Microsoft Visual C for Windows	
source file	DB2PATH\win\samples\cli\clisampl.c
command file	DB2PATH\samples\cli\clibld.bat
compile	cl -c -ALw -DDB2WIN clisampl.c
link	link /st:20000 /se:512 clisampw.obj,clisampw.exe,NUL, llibcewq+db2cliw;

Note:

- DB2PATH is the directory where DB2 is installed.
- llibcewq is the Microsoft Visual C Quick Win library used to allow ANSI C programs using stdin and stdout to run unchanged.
- DB2WIN is a macro that must be defined (equivalent to #define DB2WIN) in order to use the DB2 CLI include files.

4.3.4 DB2 CLI Functions

DB2 CLI functions can be grouped depending on which part of the application they belong to. As we said before, CLI applications can be divided into three sections: Initialization, Transaction Processing and Termination.

Table 15 shows the functions used in the Initialization section. Table 16 on page 61 shows the Transaction Processing functions, and Table 17 on page 62 shows the functions used in the Termination section.

Also, there are some other functions that can be used to retrieve information or set up the environment we are working with. Table 18 on page 62 lists these functions that can be useful to perform error handling tasks and get diagnostic information during the Transaction Processing.

Table 15. DB2 CLI Functions: Initialization

Function	Description	ODBC	X/Open	DB2
Connecting to a data source				
SQLAllocEnv	Obtains an environment handle. One environment handle is used for one or more connections.	Core	Core	v1.1
SQLAllocConnect	Obtains a connection handle.	Core	Core	v1.1
SQLConnect	Connects to a specific driver by data source name, user ID and password.	Core	Core	v1.1
SQLDriverConnect	Connect to a specific driver by connection string or optionally requests the Driver Manager and driver to display connection dialogs for the user.	Lvl 1	Lvl 1	v2.1[a]
SQLSetConnection	Sets the current active connection. Only needs to be used if you are working with Embedded SQL within a DB2 CLI application with multiple concurrent connections.	No	No	v2.1

Note:

- The DB2 column lists the first version of DB2 that supports the function.
- [a] means that runtime for this function is also available in the DB2 CAE for DOS 1.2.

Table 16 (Page 1 of 2). DB2 CLI Functions: Transaction Processing

Function	Description	ODBC	X/Open	DB2
Preparing SQL requests				
SQLAllocStmt	Allocates a statement handle	Core	Core	V1.1
SQLPrepare	Prepares an SQL statement for later execution	Core	Core	V1.1
SQLBindParameter	Assigns storage for a parameter in an SQL statement (ODBC 2.0)	Lvl 1	Lvl 1	V2.1
SQLSetParam	Assigns storage for a parameter in an SQL statement (ODBC 1.0)	Core	Core	V1.1
SQLParamOptions	Specifies the use of multiple values for parameters	Lvl 2	Lvl 2	V2.1
SQLGetCursorName	Returns the cursor name associated with the statement handle	Core	Core	V1.1
SQLSetCursorName	Specifies a cursor name	Core	Core	V1.1
Submitting requests				
SQLExecute	Executes a prepared statement	Core	Core	V1.1
SQLExecDirect	Executes a statement (not necessarily prepared)	Core	Core	V1.1
SQLNativeSQL	Returns the text of an SQL statement as translated by the driver	Lvl 2	Lvl 2	V2.1[a]
SQLNumParams	Returns the number of parameters in a statement	Lvl 2[a]	Lvl 2[a]	V2.1
SQLParamData	Used with SQLPutData() to supply parameter data at execution time	Lvl 1	Lvl 1	V2.1[a]
SQLPutData	Sends part or all of a data value for a parameter (useful for long data values)	Lvl 1	Lvl 1	V2.1[a]
Retrieving results and information about results				
SQLRowcount	Returns the number of rows affected by an INSERT, UPDATE or DELETE request	Core	Core	V1.1
SQLNumResultCols	Returns the number of columns in the result set	Core	Core	V1.1
SQLDescribeCol	Describes a column in the result set	Core	Core	V1.1
SQLColAttributes	Describes attributes of a column in the result set	Core	Core	V1.1
SQLSetColAttributes	Sets attributes of a column in the result set	No	No	V2.1
SQLBindCol	Assigns storage for a result column and specifies the data type	Core	Core	V1.1
SQLFetch	Returns a result row	Core	Core	V1.1
SQLExtendedFetch	Return multiple result rows	Lvl 2	Lvl 2	V2.1
SQLGetData	Returns part or all of one column of one row of a result set (useful for long data values)	Lvl 1	Lvl 1	V1.1
SQLMoreResults	Determines whether there are more result sets available and if so, initializes processing for the next result set	Lvl 2	Lvl 2	V2.1[a]
SQLError	Returns additional error or status information	Core	Core	V1.1
SQLGetSQLCA	Returns the SQLCA associated with a statement handle	No	No	V2.1
Large Object Support				
SQLBindFileToCol	Associates a LOB file reference with a LOB column	No	No	V2.1

Table 16 (Page 2 of 2). DB2 CLI Functions: Transaction Processing

Function	Description	ODBC	X/Open	DB2
SQLBindFileToParam	Associates a LOB file reference with a parameter marker	No	No	V2.1
SQLGetLength	Gets length of a string referenced by a LOB locator	No	No	V2.1
SQLGetPosition	Gets the position of a string within a source string referenced by a LOB locator	No	No	V2.1
SQLGetSubString	Creates a new LOB locator that references a substring within a source string (the source string is also represented by a LOB locator)	No	No	V2.1
Terminating a statement				
SQLFreeStmt	Ends statement processing and closes the associated cursor, discards pending results and optionally frees all resources associated with the statement handle	Core	Core	V1.1
SQLCancel	Cancels an SQL statement	Core	Core	V1.1
SQLTransact	Commit or rollback a transaction	Core	Core	V1.1

Note:

- The DB2 column lists the first version of DB2 that supports the function.
- [a] means that runtime for this function is also available in the DB2 CAE for DOS 1.2.

Table 17. DB2 CLI Functions: Termination

Function	Description	ODBC	X/Open	DB2
Terminating a connection				
SQLDisconnect	Closes the connection	Core	Core	V1.1
SQLFreeConnect	Releases the connection handle	Core	Core	V1.1
SQLFreeEnv	Releases the environment handle	Core	Core	V1.1

Note:

- The DB2 column lists the first version of DB2 that supports the function.

Table 18 (Page 1 of 2). DB2 CLI Functions: Information and Setup

Function	Description	ODBC	X/Open	DB2
Obtaining information about a Driver and Data Source				
SQLDataSources	Returns a list of available data sources	Lvl 2	Lvl 2	V1.1
SQLGetInfo	Returns information about a specific driver and data source	Lvl 1	Lvl 1	V1.1
SQLGetFunction	Returns supported driver functions	Lvl 1	Lvl 1	V1.1
SQLGetTypeInfo	Returns information about supported data types	Lvl 1	Lvl 1	V1.1
Setting and retrieving Driver Options				
SQLSetEnvAttr	Sets an environment option	No	No	V2.1
SQLGetEnvAttr	Returns the values of an environment option	No	No	V2.1
SQLSetConnectOption	Sets a connection option	Lvl 1	Lvl 1	V2.1[a]
SQLGetConnectOption	Returns the value of a connection option	Lvl 1	Lvl 1	V2.1[a]
SQLSetStmtOption	Sets a statement option	Lvl 1	Lvl 1	V2.1[a]

Table 18 (Page 2 of 2). DB2 CLI Functions: Information and Setup

Function	Description	ODBC	X/Open	DB2
SQLGetStmtOption	Returns a statement option	Lvl 1	Lvl 1	V2.1[a]
Obtaining information about the data sources' system tables				
SQLColumns	Returns the list of column names in specified tables	Lvl 1	Lvl 1	V2.1[a]
SQLForeignKeys	Returns a list of column names that define foreign keys for a table	Lvl 2	Lvl 2	V2.1
SQLPrimaryKeys	Returns a list of column names that define the primary key for a table	Lvl 2	Lvl 2	V2.1
SQLProcedureColumns	Returns the list of input and output parameters	Lvl 2	Lvl 2	V2.1
SQLProcedures	Returns the list of procedure names stored in a specific data source	Lvl 2	Lvl 2	V2.1
SQLSpecialColumns	Returns information about the optimal set of columns that uniquely identifies a row	Lvl 1	Lvl 1	V2.1[a]
SQLStatistics	Returns statistics about a single table and the list of its indexes	Lvl 1	Lvl 1	V2.1[a]
SQLTablePrivileges	Returns a list of tables and its privileges	Lvl 1	Lvl 2	V2.1
SQLTables	Returns a list of table names stored in a specific data source	Lvl 1	Lvl 1	V2.1[a]

Note:

- The DB2 column lists the first version of DB2 that supports the function.
- [a] means that runtime for this function is also available in the DB2 CAE for DOS 1.2.

4.4 Advanced Features

To perform special tasks while working with CLI, such as connecting to multiple databases in a distributed environment, getting information of system catalog tables, using large objects, or user-defined data types, there are some options or attributes that must be defined in the application. By changing their values, the application can change the behavior of DB2 CLI at three levels:

1. Environment

   An environment handle has attributes that affect CLI functions under the environment defined when the handle was allocated. The application can specify the value of an attribute by calling the function SQLSetEnvAttr() and can obtain current values with SQLGetEnvAttr(). You must set up the environment attributes before the connection handles have been allocated.

2. Connection

   A connection handle has attributes that affect CLI functions under the connection where they are allocated. Options can be changed considering that:

   - Some options, such as SQL_CURRENT_SCHEMA, can be set any time once the connection handle is allocated.
   - Some options, such as SQL_MAXCONN, SQL_CONNECTTYPE or SQL_SYNC_POINT., can be set only before the actual connection is established.

- Some options, like SQL_TXN_ISOLATION, can be set only after the connection is established, but while there are no outstanding transactions or open cursors.

The application can change the connection options by calling the SQLSetConnectOption() function and obtain current values with SQLGetConnectOption().

For detailed information on when each option can be changed, refer to the SQLSetConnectOption() function in the *Call Level Interface Guide and Reference - for common servers*.

3. Statement

   A statement handle has options which will affect the CLI functions that are executed using this statement handle. You should consider the following:

   - Some options can be set currently only to one specific value; this means that although there are many option values for DB2 CLI they can be set only to a specific one. That applies to SQL_ASYNC_ENABLE, which is always off and to SQL_CURSOR_TYPE, set to forward only. An option valid only in the Windows platform is SQL_QUERY_TIMEOUT.
   - Some options, such as SQL_ROWSET_SIZE, SQL_MAX_ROWS and SQL_MAX_LENGTH, can be set any time after the statement handle has been allocated.
   - Some options can only be set if there is no open cursor on the statement handle. These include SQL_CONCURRENCY, SQL_CURSOR_HOLD or SQL_NODESCRIBE.

   The application can specify the value of any option by calling the SQLSetStmtOption() function and obtain the current value with SQLGetStmtOption().

   For further information on when each option can be changed, refer to function SQLSetStmtOption() in the *Call Level Interface Guide and Reference - for common servers*.

Many applications can use the default options; however, in some situations when you need to control different environments for users of the same application, you must change them. DB2 CLI provides two ways to change the defaults at run time:

1. You can specify the default attribute value in the connecting string input to the SQLDriverConnect() function, or
2. Specify a new default attribute value in a DB2 CLI initialization file.

The DB2 CLI initialization file can change defaults only in that workstation. Default attributes specified with SQLDriverConnect() override the values in this file for that particular connection.

The following sections explain the functions and option settings needed to perform specific tasks.

4.4.1 Distributed Unit of Work

This section describes how DB2 CLI applications can be written to use coordinated distributed unit of work. That means that all commits or rollbacks to multiple database connections are coordinated.

First, you need to consider the SQL_CONNECTTYPE environment attribute, which controls how the application will operate in a coordinated or uncoordinated distributed environment. The two possible values are:

SQL_CONCURRENT_TRANS
: For multiple concurrent connections to the same database and to different databases. This is the default.

SQL_COORDINATED_TRANS
: For use with multiple databases in one transaction. This corresponds to the Type 2 CONNECT in Embedded SQL.

Also, you need to consider the SQL_SYNC_POINT option, which has the following two possible settings:

SQL_ONEPHASE
: One-phase commit is used when you want to commit the work done by each database in a multiple database transaction. This kind of transaction can only have one database updated; that is, once the first database is updated in the transaction, the other databases become read-only, and any attempt to update them within this transaction will be rejected.

SQL_TWOPHASE
: Two-phase commit enables multiple database update within a single transaction.

All connections within an application must have the same SQL_CONNECTTYPE and SQL_SYNC_POINT setting. The application should set up the environment attributes as soon as SQLAllocEnv() has been called successfully. The type of the first connection will determine the type of all subsequent connections. If an application attempts to change the connect type while there is an active connection, SQLSetEnvAttr() and SQLSetConnectOption() will return an error.

Figure 41 on page 66 shows the steps to begin multiple connections with concurrent transactions. When using multiple connections, you can choose to execute and commit all the transactions for each connection in a group as shown, or you can mix the execution of the statements for each connection and commit them at any time to end the transaction in each connection.

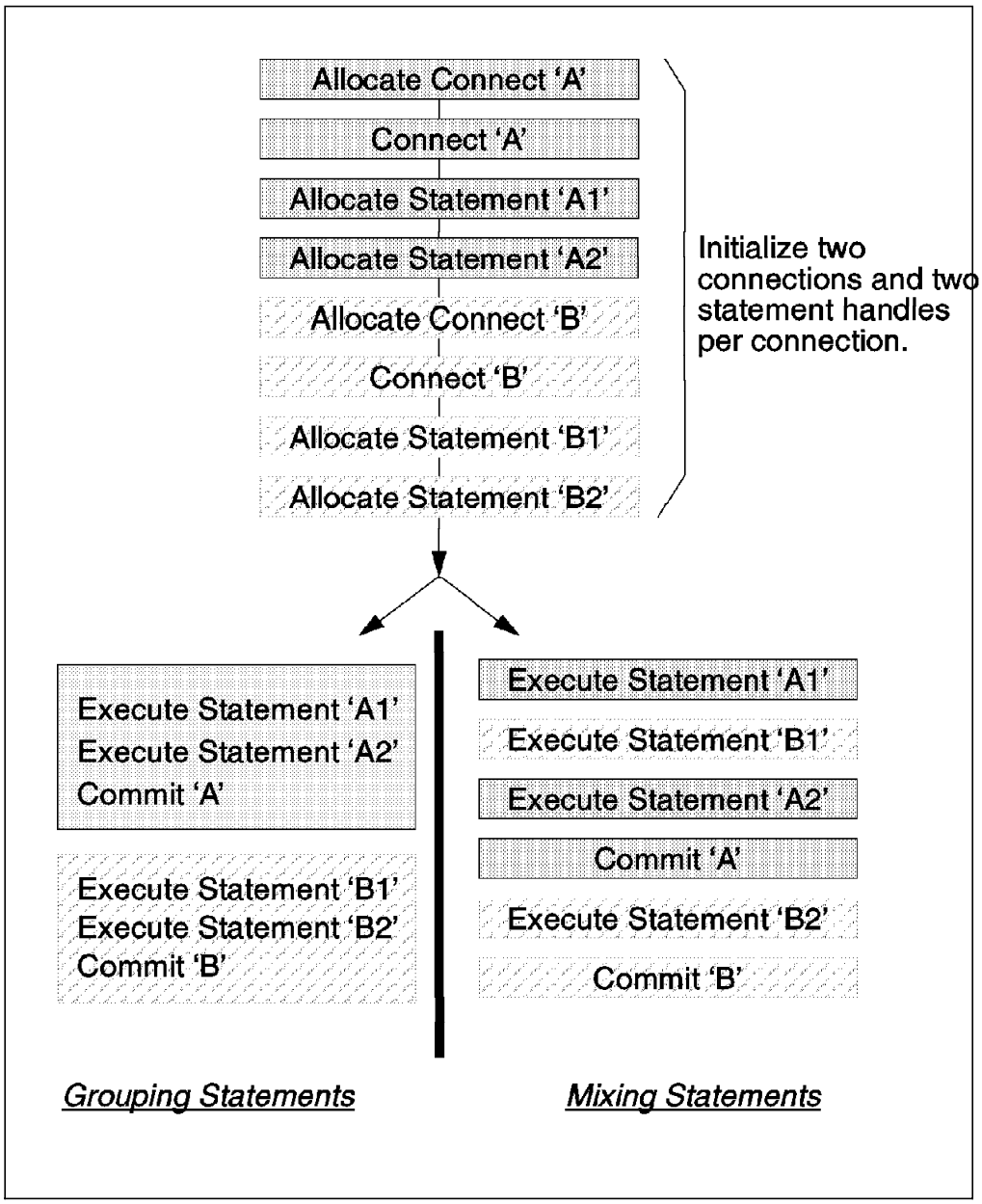


Figure 41. Multiple Connections with Concurrent Transactions

The CLI functions and attributes needed for developing applications using multiple connections are shown in Figure 42 on page 67.

```
...

rc = SQLAllocEnv(&henv);                          /* environment handle */

rc = SQLAllocConnect(henv, &hdbc[0]);      /* 1st connection */
Prompted_Connect(henv, &hdbc[0]);

rc = SQLAllocConnect(henv, &hdbc[1]);      /* 2nd connection */
Prompted_Connect(henv, &hdbc[1]);

/* Transaction Processing: allocate and execute statements */
...

rc = SQLDisconnect(hdbc&hdbc[0]);         /* release handles */
rc = SQLDisconnect(hdbc&hdbc[1]);
rc = SQLFreeConnect(hdbc&hdbc[0]);
rc = SQLFreeConnect(hdbc&hdbc[1]);
rc = SQLFreeEnv(henv);
...
```

Figure 42. DB2 CLI Functions needed for Multiple Connections

Figure 43 on page 68 shows the steps to begin multiple connections with coordinated transactions. A coordinated transaction means that you must execute all the statements for each connection before you perform the commit. You can choose to execute all the statements corresponding to one connection and then continue with the other connection statements, or execute the statements for both connections in any order and then perform the commit.

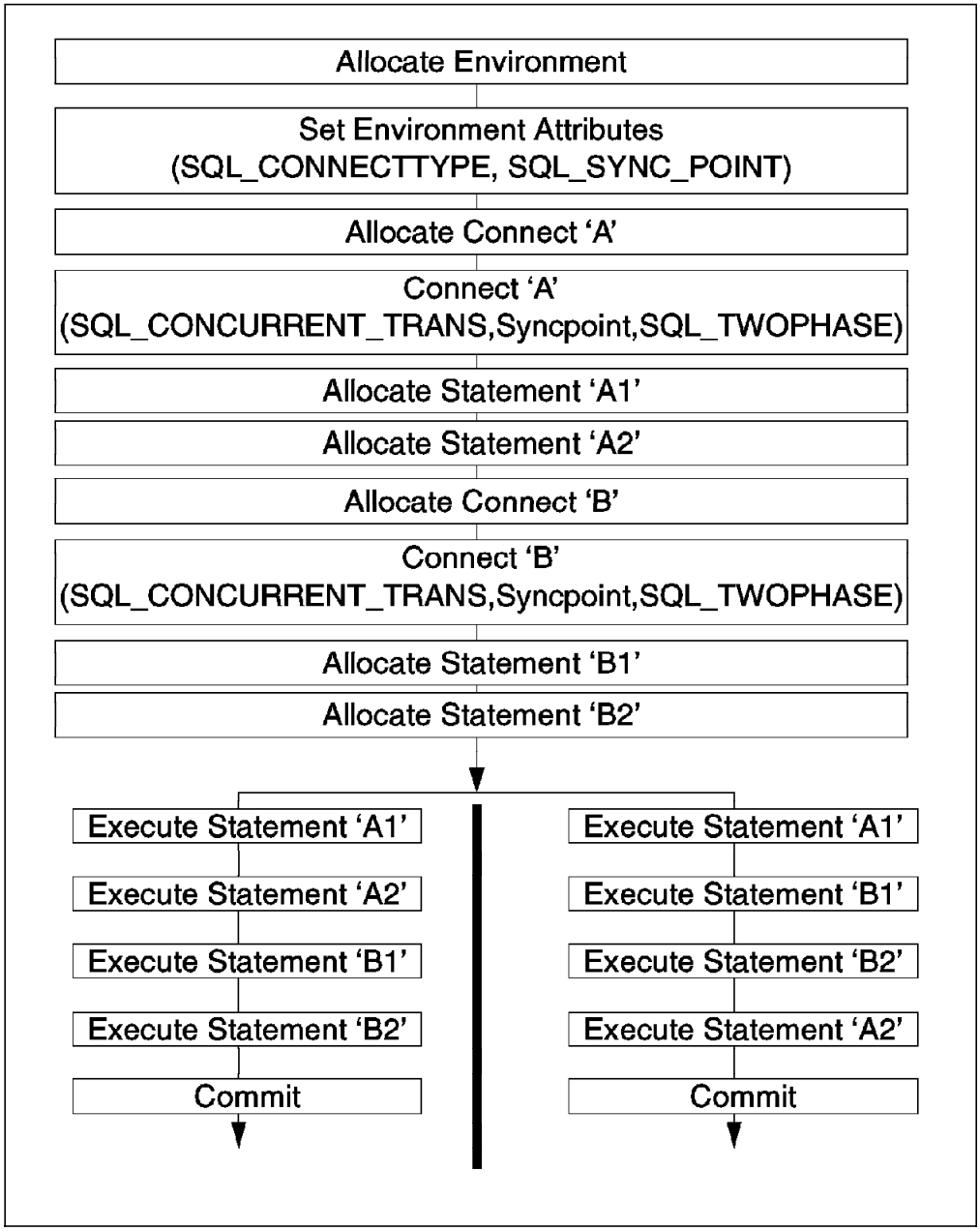


Figure 43. Multiple Connections with Coordinated Transactions

The environment and connection handle functions used for coordinated transactions are shown in Figure 44 on page 69. Note that in coordinated transactions, it is recommended that you define both `SQL_CONNECTTYPE` and `SQL_SYNC_POINT` values in the application.

```
rc = SQLAllocEnv(&henv);                          /* environment handle  */

rc = SQLSetEnvAttr(henv, SQL_CONNECTTYPE,         /* set SQL_CONNECTTYPE */
       (SQLPOINTER) SQL_COORDINATED_TRANS, 0);
rc = SQLSetEnvAttr(henv, SQL_SYNCPOINT,           /* set SQL_SYNCPOINT   */
       (SQLPOINTER) SQL_ONEPHASE, 0);

rc = SQLAllocConnect(henv, &hdbc[0]);      /* 1st connection */
Prompted_Connect(henv, &hdbc&hdbc[0]);
rc = SQLAllocConnect(henv, &hdbc&hdbc[1]); /* 2nd connection */
Prompted_Connect(henv, &hdbc&hdbc[1]);

/* Transaction Processing: allocate and execute statements */
...

rc = SQLDisconnect(hdbc&hdbc[0]);         /* release handles */
rc = SQLDisconnect(hdbc&hdbc[1]);
rc = SQLFreeConnect(hdbc&hdbc[0]);
rc = SQLFreeConnect(hdbc&hdbc[1]);
rc = SQLFreeEnv(henv);
```

Figure 44. DB2 CLI Functions Required for Coordinated Transactions

4.4.2 Querying Catalog Tables

In some applications, you might need to display a list of tables or column names for the select statement to work with. To do so, the application can call the DB2 CLI catalog functions.

These catalog functions provide a generic interface to issue queries and return consistent result sets through a statement handle. This is conceptually equivalent to using SQLExecDirect() to execute a select against the system catalog tables.

After calling these functions, the application can fetch individual rows by using the function SQLFetch(). Because some of the catalog functions may result in very complex queries, it is recommended that you save the information returned rather than making repeated calls to get the same information.

Table 19 lists DB2 CLI catalog functions and their usage. For further information, refer to the *Call Level Interface Guide and Reference Book*.

Table 19 (Page 1 of 3). DB2 Call Level Interface Catalog Functions

Function	Purpose	Usage
SQLColumnPrivileges	Returns a list of columns and associated privileges for the specified table.	An application may call this function after a call to SQLColumns() to determine column privilege information. It can use the character strings returned in the TABLE_SCHEM, TABLE_NAME and COLUMN_NAME columns as input arguments to this function.

Table 19 (Page 2 of 3). DB2 Call Level Interface Catalog Functions

Function	Purpose	Usage
SQLColumns	Returns a list of columns in the specified tables. The information returned can be retrieved using the same functions to fetch the results of a query.	An application may call this function after a call to SQLTables() . It should use the character strings in the TABLE_SCHEMA and TABLE_NAME columns as input values. This function doesn't return information on the columns in a result set, so SQLDescribeCol() or SQLColAttributes() should be used.
SQLForeignKeys	Returns information about foreign keys for the specified tables.	Can return a result set containing: • If szPkTableName contains a table name and szFkTableName is empty, then the primary key of a specified table and all the foreign keys that refer to it is returned. • If szFkTableName contains a table name and szPkTableName is empty, then all the foreign keys in the specified table and the primary keys to which they refer will be returned. • If both szFkTableName and szPkTableName contain table names, then the foreign keys in the table specified in szFkTableName that refer to the primary key of the table specified in szPkTableName is returned. This should be one key at the most.
SQLPrimaryKeys	Returns a list of column names that define the primary key for a table.	It will return the primary key columns from a single table. Search patterns cannot be used to specify the schema qualifier or the table name. If the specified table does not contain a primary key, an empty result set is returned.
SQLProcedureColumns	Returns a list of input and output parameters associated with a procedure.	To return such a list, the pseudo catalog table for stored procedure registration must have been created and populated. For further information on this pseudo catalog table, refer to *Pseudo Catalog Table for Stored Procedure Registration* in the *Call Level Interface Guide and Reference Book*. For versions of DB2 servers that do not provide facilities for a stored procedure catalog, an empty result set will be returned.
SQLProcedures	Returns a list of procedure names that have been registered at the server and match the specified search pattern.	The same as the SQLProcedureColumns() function.
SQLSpecialColumns	Returns unique row identifier information (primary key or unique index) from a table.	DB2 CLI will return the best row identifier column set based on its own criterion (there are many ways to uniquely identify any row in a table). If there is no column set to uniquely identify a row, an empty result set is returned.

Table 19 (Page 3 of 3). DB2 Call Level Interface Catalog Functions

Function	Purpose	Usage
SQLStatistics	Retrieves index information for a given table. It also returns the cardinality and the number of pages associated with the table and the indexes on the table.	This function returns two types of information: • Statistics information for the table (if available): When TYPE column is set to SQL_TABLE_STAT, returns the number of rows in the table and number of pages used to store the table. When the TYPE column indicates an index, returns the number of unique values in the index and the number of pages used to store the indexes. • Information about each index, where each index column is represented by one row of the result set.
SQLTablePrivileges	Returns a list of tables and associated privileges for each table.	If multiple privileges are associated with any given table, each privilege is returned as a separate row. The granularity of each privilege reported may or may not apply at the column level. For other data sources, the application must call SQLColumnPrivileges() to discover if the individual columns have the same table privileges.
SQLTables	Returns a list of table names and associated information stored in the system catalog of the connected data source.	To determine the type of access permitted, the application can call SQLTablePrivileges(). Otherwise, the application must be able to handle a situation where the user selects a table for which SELECT privileges are not granted.

The catalog table's columns are defined in a specified order, and in future releases, other columns may be added to the end of each defined result set. Applications should be written in a way that will not be affected by these changes.

Catalog functions have input arguments that are used to either identify or constrain the amount of information to be returned. CatalogName must always be a null pointer, with its length set to 0, because DB2 CLI does not support three-part naming.

You have to be careful in passing input to catalog functions. The following needs to be considered:

- If treated as ordinary arguments, input will be taken literally and is case sensitive. The input treated as an ordinary argument identifies the information desired; so an error results if a null pointer is passed.
- If treated as pattern-values, the input will be used to constrain the size of the result set by including only matching rows for the where clause. If the application passes a null pointer, there is no restriction for the query.

  If a catalog function has more than one pattern-value input argument, they are treated as where clauses joined by AND; so the rows shown are only the ones that meet all the conditions.

An example of using catalog functions for queries can be found in 4.5.1, " Querying Catalog Tables" on page 86.

4.4.3 Using Arrays

When you need to insert, delete or change multiple fields in a data entry form and send them to the database, DB2 CLI provides an array input method. This method lets you update all the rows or columns for an INSERT, DELETE, or UPDATE by using a single SQLExecute() call.

4.4.3.1 Input Parameters

Using the array method involves the binding of parameter markers to arrays of storage locations with the SQLBindParameter() call. For character and binary input data, the application uses the maximum input buffer size argument, cbValueMax, on the SQLBindParameter() call to indicate to DB2 CLI the location of values in the input array. For other input data types, the length of each element is assumed to be the size of the C data type (See Table 8 on page 50).

The SQLParamOptions() function is called to specify how many elements are in the array before the execution of the SQL statement.

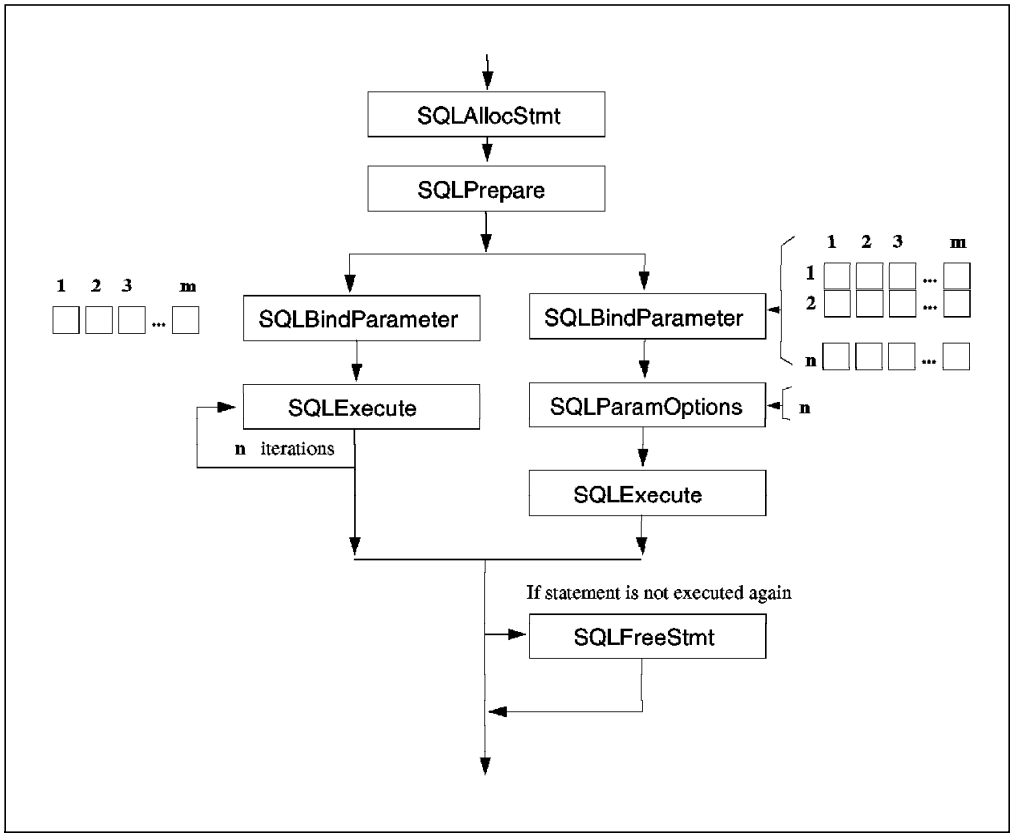


Figure 45. Array Method for Input Parameters

Figure 45 shows two methods of executing a statement with ***m*** parameters ***n*** times. The method shown on the left calls SQLExecute() once for each set of parameter values. The array method on the right calls SQLParamOptions() to specify the number of rows *(n)* and then calls SQLExecute() only once. Figure 46 on page 73 shows the CLI functions used to insert values using arrays of data.

```
...

/* SQL statement */
   SQLCHAR    stmt[] =
     "INSERT INTO product VALUES (?, ?, ?);

/* Input arrays  */
   SQLINTEGER Prod_Num [NUM_PR] = {
              100, 200, 300, 400, 500 };

   SQLCHAR    Desc [NUM_PR][60] = {
              "Product 100", "Product 200", "Product 300",
              "Product 400", "Product 500" };

   SQLDOUBLE  Price [NUM_PR] = {
              50.15, 25.30, 63.25, 28.50, 45.90 };

/* Initialization Section */
   rc = SQLAllocEnv(&henv);
   rc = DBconnect(henv, &hdbc);

/* Transaction Processing Section */
   rc = SQLAllocStmt(hdbc, &hstmt);
   rc = SQLPrepare(hstmt, stmt, SQL_NTS);

   /* Binding parameters */
   rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT,
          SQL_C_LONG, SQL_INTEGER, 0, 0, Prod_Num, 0, NULL);
   rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT,
          SQL_C_CHAR, SQL_VARCHAR, 60, 0, Desc, 60, NULL);
   rc = SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT,
          SQL_C_DOUBLE, SQL_DECIMAL,10, 2, Price, 0, NULL);
   rc = SQLParamOptions(hstmt, NUM_PR, &pirow);

   /* Execute and commit transaction */
   rc = SQLExecute(hstmt);
   rc = SQLFreeStmt(hstmt, SQL_DROP);
   rc = SQLTransact(henv, hdbc, SQL_COMMIT);

/* Termination Section */
   rc = SQLDisconnect(hdbc);
   rc = SQLFreeConnect(hdbc);
   rc = SQLFreeEnv(henv);
...
```

Figure 46. DB2 CLI Functions Using Arrays for Input Parameters

Note that both methods must call SQLBindParameter() once for each parameter (*m*). In case of character or binary data, there is no method to specify the size of each element in the array.

SQLSetParam() must not be used to bind an array storage location to a parameter marker. The function SQLBindParameter() should be used to perform the binding.

For queries with parameter markers on the WHERE clause, an array of input values will cause multiple sequential result sets; each one can be processed before moving on to the next one by calling SQLMoreResults().

4.4.3.2 Retrieving a Result Set

Other applications may need to issue a query statement and then execute SQLFetch() on each row of the result set to move them into application variables. These variables need to have been bound using SQLBindCol() before this can be done. Each column or row of the result set that you wish to store must be fetched into the application. Before you are able to perform the fetch, the variables you are using must be bound.

This can be done by retrieving multiple rows of data (a rowset) at a time into an array, instead. SQLBindCol() also assigns storage for application array variables.

Figure 47 shows ***m*** columns bound. This means that ***m*** calls to SQLBindCol() are required. You may you select to do ***n*** iterations of SQLFetch() or a single SQLExtendedFetch() call.

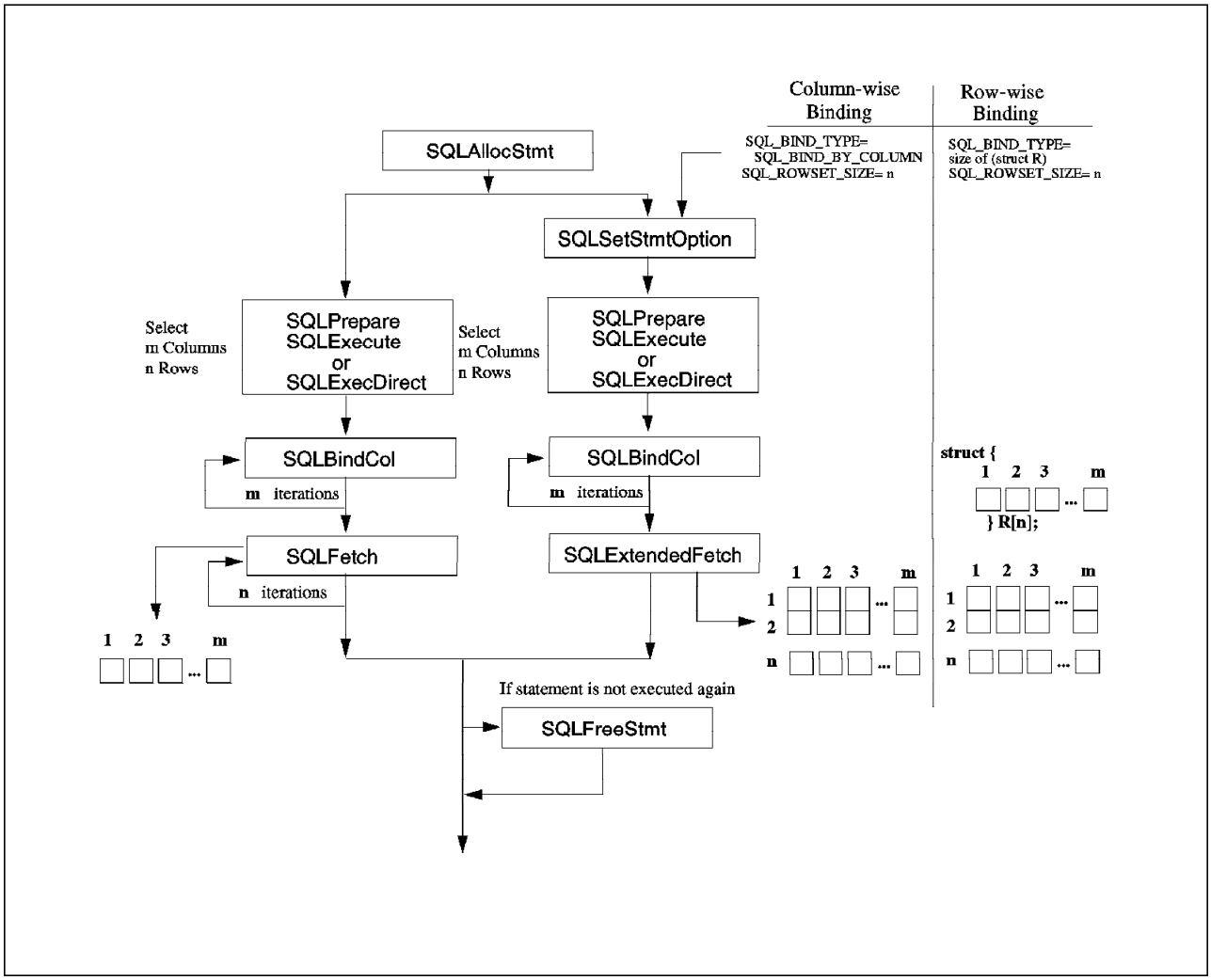


Figure 47. Array Method to Retrieve Data

By default, the binding of rows is by column; that is, column-wise. This is symmetrical to using SQLBindParameter() on the input parameters.

- The first column at the right side of Figure 47 shows the function flows to column-wise retrieval.

To work with column-wise retrieval, you must define variables for each column to be retrieved as an array of data.

After allocating the statement, the application calls SQLSetStmtOption() with the SQL_ROWSET_SIZE attribute to indicate how many rows to retrieve at a time. When this value is greater than 1, DB2 CLI knows to treat the deferred output data pointer and length pointer as pointers to arrays of data and its length.

The application calls SQLExtendedFetch() to retrieve ROWSET_SIZE rows of data at a time. DB2 CLI uses the maximum buffer size argument, cbValueMax, on the SQLBindCol() function to determine where to store successive rows of data in the array. If the number of rows in the result set is greater than the SQL_ROWSET_SIZE value, you must perform multiple calls to SQLExtendedFetch().

Figure 48 shows the definition of the array structure and the functions used to retrieve data in column-wise retrieval.

```
/* SQL statement */
   SQLCHAR       stmt[] =
     "SELECT deptnumb, deptname, id, name FROM staff, org \
             WHERE dept=deptnumb AND job = 'Mgr'";
   SQLINTEGER    deptnumb[ROWSET_SIZE];

/* Variables definition */
   SQLCHAR       deptname[ROWSET_SIZE][15];
   SQLINTEGER    deptname_l[ROWSET_SIZE];
   SQLSMALLINT   id[ROWSET_SIZE];
   SQLCHAR       name[ROWSET_SIZE][10];
   SQLINTEGER    name_l[ROWSET_SIZE];

/* Initialization section */
   ...

/* Transaction Processing */
   rc = SQLAllocStmt(hdbc, &hstmt);
   rc = SQLSetStmtOption(hstmt, SQL_ROWSET_SIZE, ROWSET_SIZE);
   rc = SQLExecDirect(hstmt, stmt, SQL_NTS);

   /* Binding columns */
   rc = SQLBindCol(hstmt, 1, SQL_C_LONG, (SQLPOINTER) deptnumb, 0, NULL);
   rc = SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER)
                  deptname, 15, deptname_l);
   rc = SQLBindCol(hstmt, 3, SQL_C_SSHORT, (SQLPOINTER) id, 0, NULL);
   rc = SQLBindCol(hstmt, 4, SQL_C_CHAR, (SQLPOINTER) name, 10, name_l);

   /* Execute SQL */
   while ((rc = SQLExtendedFetch(hstmt, SQL_FETCH_NEXT, 0,
                &pcrow, Row_Stat)) == SQL_SUCCESS) {
         for (i = 0; i < pcrow; i++) {
         printf("%8ld %-14s %8d %-9s\n", deptnumb[i],
              deptname[i], id[i], name[i]);
         {
         if (pcrow < ROWSET_SIZE)
             break;
         }
   rc = SQLFreeStmt(hstmt, SQL_DROP);
   rc = SQLTransact(henv, hdbc, SQL_COMMIT);

/* Termination section */
   ...
```

Figure 48. DB2 CLI Function Needed for Column-Wise Retrieval

- The second column at the right side of Figure 47 on page 74 shows the function flows to row-wise retrieval.

  The row-wise binding associates an entire row of the result set with a structure. This structure is defined as an array of data columns and their respective lengths.

  The application needs to call SQLSetStmtOption() with the SQL_ROWSET_SIZE attribute to indicate how many rows will be retrieved at a time. It must use the SQL_BIND_TYPE attribute to set the size of the structure.

  DB2 CLI treats the deferred output data pointer, SQLBindCol(), as the address of the data field for the column in the first element of the array and the deferred output length pointer as the address of the associated length field of the column.

  To retrieve the data, DB2 CLI uses the structure size provided with the SQL_BIND_SIZE attribute to determine where to store successive rows in the array of structures.

  Figure 49 on page 77 shows the definition of the array structure and the functions used to retrieve data in row-wise retrieval.

```
/* SQL statement */
   SQLCHAR       stmt[]
     "SELECT deptnumb, deptname, id, name FROM staff, org \
             WHERE dept=deptnumb AND job = 'Mgr'";

/* Array definition */
   struct {
     SQLINTEGER      deptnumb_l; /* length */
     SQLINTEGER      deptnumb; /* value  */
     SQLINTEGER      deptname_l;
     SQLCHAR         deptname[15];
     SQLINTEGER      id_l;
     SQLSMALLINT     id;
     SQLINTEGER      name_l;
     SQLCHAR         name[10];
          } R[ROWSET_SIZE];
     SQLUSMALLINT    Row_Stat[ROWSET_SIZE];
     SQLUINTEGER     pcrow;
     int             i;

/* Initialization section */
   ...

/* Transaction Processing */
   rc = SQLAllocStmt(hdbc, &hstmt);
   rc = SQLSetStmtOption(hstmt, SQL_ROWSET_SIZE, ROWSET_SIZE);
   rc = SQLSetStmtOption(hstmt, SQL_BIND_TYPE, sizeof(R)/ ROWSET_SIZE);
   rc = SQLExecDirect(hstmt, stmt, SQL_NTS);

   /* Binding columns   */
   rc = SQLBindCol(hstmt, 1, SQL_C_LONG, (SQLPOINTER)
          &R[0].deptnumb, 0,&R[0].deptnumb_l     );
   rc = SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER)
          R[0].deptname, 15, &R[0].deptname_l);
   rc = SQLBindCol(hstmt, 3, SQL_C_SSHORT, (SQLPOINTER)
          &R[0].id, 0, &R[0].id_l);
   rc = SQLBindCol(hstmt, 4, SQL_C_CHAR, (SQLPOINTER)
          R[0].name, 10, &R[0].name_l);

   /* Execute SQL */
   while ((rc = SQLExtendedFetch(hstmt, SQL_FETCH_NEXT, 0,
                &pcrow, Row_Stat))  == SQL_SUCCESS) {
         for (i = 0; i < pcrow; i++) {
         printf("%8ld %-14s %8d %-9s\n", R[i].deptnumb,
         R[i].deptname,R[i].id,R[i].name);
         {
         if (pcrow < ROWSET_SIZE)
            break;
         }
   rc = SQLFreeStmt(hstmt, SQL_DROP);
   rc = SQLTransact(henv, hdbc, SQL_COMMIT);

/* Termination section */
   ...
```

Figure 49. DB2 CLI Functions Needed for Row-Wise Retrieval

4.4.4 Using Compound SQL

Another way to execute a series of INSERT, UPDATE and DELETE statements in an efficient way is to use compound SQL. This allows you to group multiple statements into a single group that can be executed in a single continuous stream, which reduces network traffic and execution time.

Statements within a compound SQL statement are called sub-statements and may be any SQL statement that can be prepared dynamically.

Compound SQL statements cannot be nested. There must not be any dependency between the group of statements in the compound SQL. Also the correct authorization must be available for all of the individual sub-statements.

A group of compound SQL statements is specified by surrounding these sub-statements with a BEGIN COMPOUND statement and an END COMPOUND statement. Using DB2 CLI, a BEGIN COMPOUND statement is called by the SQLExecDirect() function. Then, SQLExecute() processes the sub-statements, and finally another SQLExecDirect() calls the END COMPOUND statement.

There are some parameters you can use with BEGIN COMPOUND statement to define its behavior:

ATOMIC If any of the sub-statements fail, then all the changes made to the database are undone. This is not supported in DRDA environments.

NOT ATOMIC Regardless of the failure of any sub-statement, all the changes made by successful sub-statements in the database will not be undone.

STATIC Specifies that the input variables for all sub-statements retain their original value. If one variable is set by more than one sub-statement, then its value following the compound SQL statement is set by the last sub-statement.

STOP AFTER FIRST *n* STATEMENTS
Only *n* sub-statements will be executed. If omitted, all the sub-statements are executed.

To finish the compound SQL statement, the END COMPOUND statement can specify the following option:

COMMIT This option will commit all the sub-statements if they are executed successfully. The commit applies to the current transaction, including statements that precede the compound statement.

In a coordinated distributed connection, where the compound SQL is part of the transaction, if COMMIT is specified, an error with SQLSTATE 25000 will be returned (refer to 4.4.1, " Distributed Unit of Work" on page 65). If COMMIT is not specified after END COMPOUND, the sub-statements will not be committed unless the application is operating on auto-commit mode, in which case the commit is issued at the END COMPOUND statement. See AUTOCOMMIT in Table 11 on page 54.

Figure 50 on page 79 shows the sequence of function calls needed to execute a compound SQL statement.

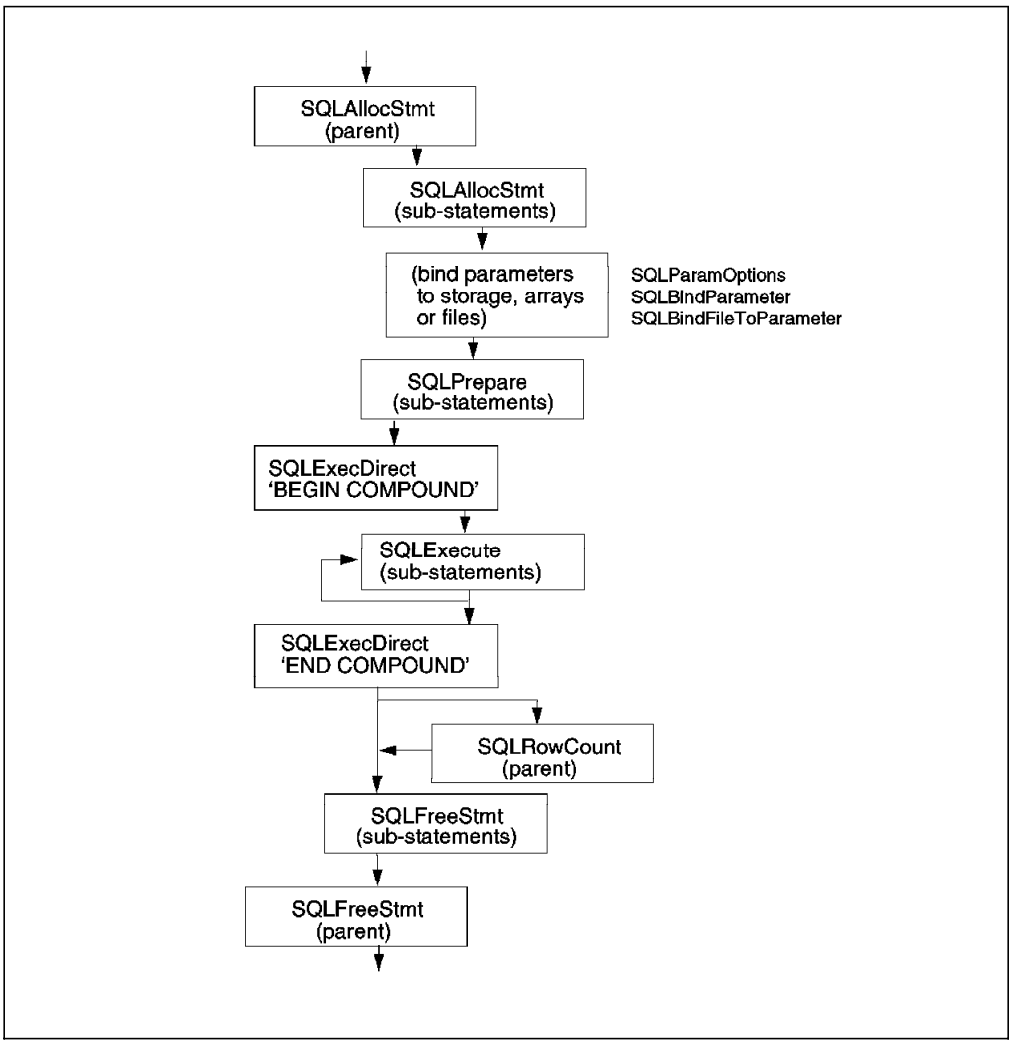


Figure 50. Compound SQL Statement Structure

Figure 51 on page 80 shows the CLI functions used to execute a compound SQL with two sub-statements.

```
...

/* Handles and sub-statements definition */
   SQLHSTMT    hstmt, cmhstmt[2];
   SQLCHAR     compst[2][512] = {
     "INSERT INTO awards (id, award) "
            "SELECT id, 'Sales Merit' from staff "
                  "WHERE job = 'Sales' AND (comm/100 > years)",
     "INSERT INTO awards (id, award) "
            "SELECT id, 'Clerk Merit' from staff "
                  "WHERE job = 'Clerk' AND (comm/50 > years)",
              };

/* Initialization section */
   ...

/* Transaction Processing Section */
   rc = SQLAllocStmt(hdbc, &hstmt);

   /* Prepare 2 substatements */
   for (i = 1; i < 2; i++) {
   rc = SQLAllocStmt(hdbc, &cmhstmt[i]);
   rc = SQLPrepare(cmhstmt[i], compst[i], SQL_NTS);
   rc = SQLExecDirect(hstmt, (SQLCHAR *)
          "BEGIN COMPOUND NOT ATOMIC STATIC", SQL_NTS);

   /* Execute 2 substatements and commit transaction */
   for (i = 1; i < 2; i++) {
   rc = SQLExecute(cmhstmt[i]);
   rc = SQLExecDirect(hstmt, (SQLCHAR *)
          "END COMPOUND COMMIT", SQL_NTS);
   rc = SQLFreeStmt(hstmt, SQL_DROP);
   for (i = 1; i < 2; i++) {
   rc = SQLFreeStmt(cmhstmt[i], SQL_DROP);
        }
   rc = SQLTransact(henv, hdbc, SQL_COMMIT);

/* Termination section */
...
```

Figure 51. DB2 CLI Functions Needed for Compound SQL

When using compound SQL, it is important to remember the following:

- The BEGIN COMPOUND and END COMPOUND statements are executed with the same statement handle.
- Each sub-statement must have its own statement handle.
- All statement handles must belong to the same connection and have the same isolation level.
- The sub-statements should be prepared before the BEGIN COMPOUND statement. That is not necessary in DRDA environments where some optimization may reduce network flow.
- The statement handles must remain allocated until the END COMPOUND statement is executed.
- The only functions that may be called using the statement handles allocated for the compound sub-statements are:
  - SQLAllocStmt()
  - SQLSetParam()

  - SQLBindParameter()
  - SQLParamOptions()
  - SQLBindFileToParam()
  - SQLParamData()
  - SQLExecDirect()
  - SQLPrepare()
  - SQLExecute()
- SQLTransact() cannot be called for the same connection, or for any connect requests between BEGIN and END COMPOUND, because commit or rollback is done at the moment END COMPOUND is called.
- The sub-statements may be executed in any order.
- SQLRowCount() or SQLGetSQLCA() can be called using the same statement handle as the BEGIN and END COMPOUND statement to get an aggregate count of the rows affected.

4.4.5 Large Objects

There are three LOB data types currently supported by DB2:

- Binary Large Object (BLOB)
- Character Large Object (CLOB)
- Double-Byte Character Large Object (DBCLOB)

These data types are represented symbolically as SQL_BLOB, SQL_CLOB and SQL_DBCLOB, respectively. For the default C symbolic name, refer to Table 8 on page 50.

Managing information contained in large objects within an application could be considered as requiring large amounts of resource. This is not true with the correct use of DB2 CLI functions when combined with other concepts such as locators and direct file input and output.

4.4.5.1 LOB Locators

A LOB (Large Object) locator is a runtime concept, a mechanism that allows an application to manipulate a large object value in an efficient, random-access fashion. It is not a persistent type and is not stored in the database; it only exists within the transaction in which it was created.

A LOB locator is a simple token value that represents a LOB value. It is not a reference to a column in a row, and there is no operation that could be performed on a locator that would affect the original LOB value stored in the row.

An application can retrieve a LOB locator into an application variable using the SQLBindCol() or SQLGetData() functions, and then it can apply the following DB2 CLI functions to the associated LOB value via the locator:

SQLGetLength() Gets the length of a string that is represented by a LOB locator.

SQLGetPosition() Gets the position of a search string within a source string represented by a LOB locator. The search string can also be represented by a LOB locator.

There are three ways in which the locators are implicitly allocated:

- Fetching a bound LOB column to the appropriate C locator type.
- Calling SQLGetSubString() and specifying that the substring be retrieved as a locator.
- Calling SQLGetData() on an unbound LOB column and specifying the appropriate C locator type. The C locator type must match the LOB column type or an error will occur.

The locator can be explicitly freed before the end of the transaction by the FREE LOCATOR statement which may be called by using the SQLExecDirect() function.

4.4.5.2 Direct File Input and Output

In other cases, when the application needs the entire LOB column value, you can use direct file input and output for LOBs. The two DB2 CLI LOB file access functions are:

SQLBindFileToCol() Associates a LOB column in a result set with a file name.

SQLBindFileToParam() Associates a LOB parameter marker with a file name.

The file name can be a completed path name (recommended) or a relative file name, in which case the file is appended to the current path of the client process.

Use of SQLBindFileToParam() is more efficient than the sequential input of data segments using SQLPutData() because SQLPutData() uses a temporary file for data segments before using the SQLBindFileToParam() technique to send the LOB data value to the server.

Figure 52 on page 83 shows how to retrieve a character LOB.

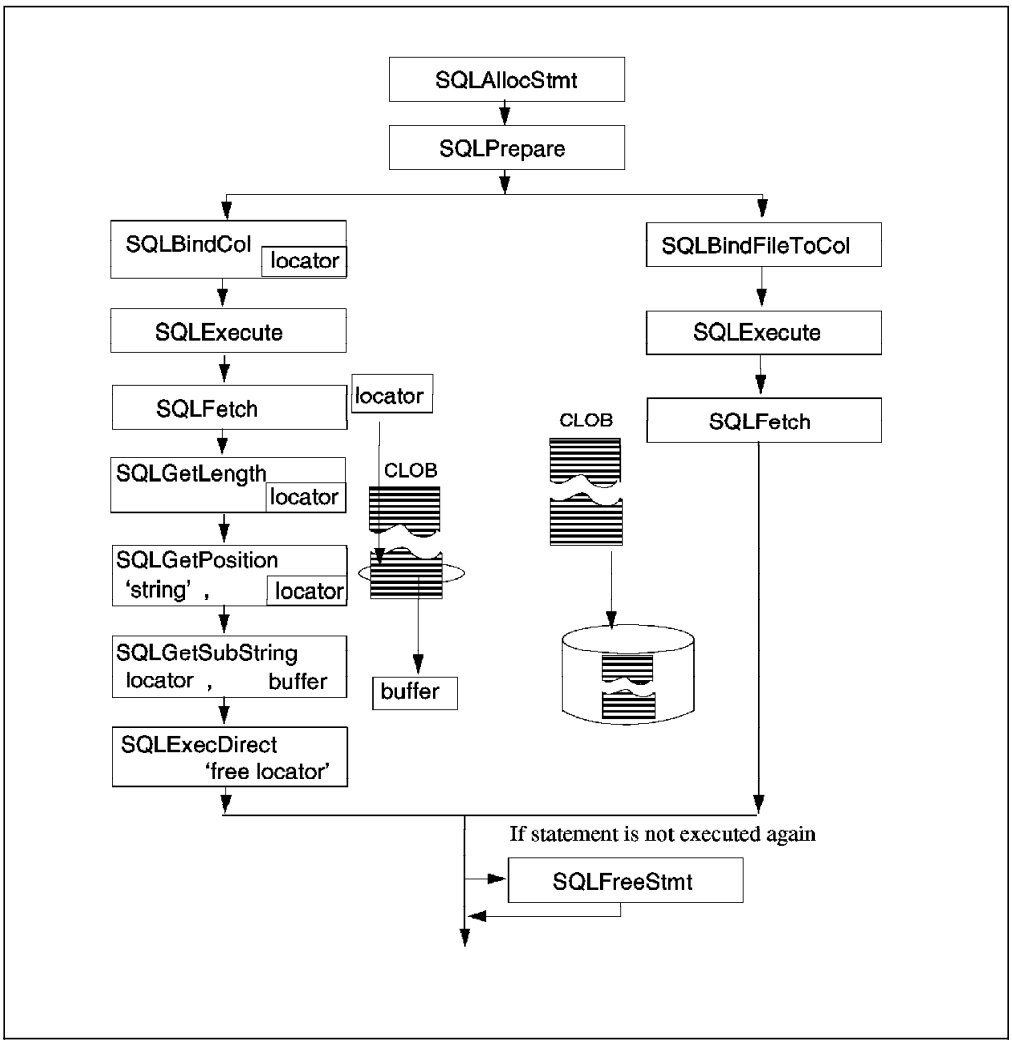


Figure 52. Using LOBS with Locators and Files

The left side of Figure 52 shows the use of a locator to extract a character string from the CLOB column without transferring the entire CLOB to an application buffer. A LOB locator is fetched for use as an input parameter to search a substring in the CLOB (See 4.5.2, "Using CLOB Locator" on page 88).

The right side of Figure 52 shows how to fetch a CLOB column into a file. The file is bound to the CLOB column; so when the row is fetched, the entire CLOB value is transferred directly to this file.

Not all DB2 servers have Large Object support. To determine if any of the LOB functions are supported, call `SQLGetFunctions()` with the appropriate function name argument value.

4.4.6 User-Defined Types (UDTs)

DB2 Version 2 supports the use of User-Defined Types (UDTs), which means users can define new distinct data types. There are some rules on how DB2 CLI handles with the UDTs as the CLI functions can only deal with the built-in data types.

Data Conversion

DB2 CLI will consider the UDTs as its source data type; that is, the data type used to define the UDT. So all the SQL to C data type conversion which is listed in Table 8 on page 50 is applied to UDTs. A UDT will have the same default C type as its source, built-in type.

For example, users define the following UDT:

```
CREATE DISTINCT TYPE CNUM AS INTEGER WITH COMPARISON
```

Data conversion will convert the CNUM UDT into SQL_C_LONG data type which is the default C type for integer.

Column Information

When you use the SQLDescribeCol() function in your application, it will return the information of the built-in type, not the UDT type.

You can obtain the UDT name by calling SQLColAttributes() with SQL_COLUMN_DISTINCT_TYPE as the input descriptor argument.

Parameter Marker in a WHERE clause

If you are going to use a parameter marker for the UDT in an SQL statement with a WHERE clause, you have to use the CAST function in the SQL statement. The CAST function is used to cast either the parameter marker or the UDT to a different data type.

Below is an example of using the CAST function with User-Defined Types.

The following table, CUSTOMER, is using the UDT defined above:

```
CREATE TABLE CUSTOMER (
       Cust_Num      CNUM NOT NULL,
       First_Name    CHAR(30) NOT NULL,
       Last_Name     CHAR(30) NOT NULL,
       PRIMARY KEY   (Cust_Num) )
```

The parameter marker cannot take a CNUM data type; so this SQL statement will not work since the comparison fails due to incompatible data types:

```
SELECT first_name FROM customer WHERE cust_num = ?
```

The right SQL statement should be one of the following:

- Casting the column using CNUM to integer:

  ```
  SELECT first_name FROM customer WHERE cast(cust_num as integer) = ?
  ```

- Casting the parameter marker to integer:

  ```
  SELECT first_name FROM customer WHERE cust_num = cast(? as integer)
  ```

4.4.7 Stored Procedures

When working in client/server environments, an application can be designed to run in two parts: one part on the client and the other part on the server. The part that runs on the server is called the stored procedure and can be written using Embedded SQL or DB2 CLI functions.

Some advantages of using stored procedures are:

- Avoids network transfer of large amounts of data obtained as intermediate results in a sequence of queries.
- Deploys client database applications into client/server pieces.
- If written in Embedded static SQL:
  1. Performance

     As static SQL is prepared at precompile time, there is no overhead of access plan (package) generation.

  2. Encapsulation

     Users do not need to know details about the database objects to access them.

  3. Security

     Users' access privileges are encapsulated in the package(s) associated with the stored procedure(s). You don't need to grant explicit access to each database object.

To invoke a stored procedure from a DB2 CLI application, you have to pass the following CALL statement syntax to SQLExecDirect() or to SQLPrepare() followed by SQLExecute():

```
CALL procedure-name (?)
```

Where procedure-name is the name of the procedure that will be executed in the DB2 common server. It can take the following forms:

procedure-name
: The stored procedure name with no extension. The server assumes that the function routine name to be executed is identical to the library name, and that it is in the sqllib/function directory, by default.

procedure-name!func-name
: You can use exclamation mark (!) to specify a library name identified by *procedure-name* and the function to be executed. This allows similar function routines to be in the same stored procedure library.

/u/db2user/procedure-name!func-name
: You can specify the stored procedure library as a full path name.

(?)
: This means that arguments for the stored procedure must be passed using parameter markers. Literals can be used if the vendor escape call statement is used.

The parameter markers in the call statement are bound to application variables by using SQLBindParameter(). In order to avoid unnecessary flow of data between client and server, the application should specify the parameter type of input arguments as SQL_PARAM_IN and output arguments as SQL_PARAM_OUTPUT. Arguments used for both input and output have a parameter type of SQL_PARAM_INPUT_OUTPUT.

Stored procedures can also be invoked by using the Database Application Remote Interface (DARI) API from clients of any DB2 server, but it can only access stored procedures on DB2 for Common Server platforms, not those on Distributed Relational Database Architecture (DRDA) servers.

Although stored procedures written in Embedded SQL offer more advantages, you may wish to move components of the DB2 CLI application to run on the server, of course implemented as stored procedures written using DB2 CLI.

For further information, refer to *CALL* statement in the *DB2 SQL Reference Book*. A good example of using stored procedures is included in the samples/cli directory as outsrv.c and outcli.c files.

4.5 Examples

In this section, we explain some examples of DB2 CLI applications. You need to compile and link the C programs you have written as mentioned in 4.3.3, " Compiling and Linking Applications" on page 59.

4.5.1 Querying Catalog Tables

An example of a DB2 CLI application that lists the columns for a table is shown in Figure 53 on page 87. In this example, we are using the `SQLColumns()` function to get the column information instead of using an SQL query. The result is returned in an SQL result set; so we can use the same functions used to retrieve the result set generated by a query.

The catalog functions are used to obtain information about the data source's system tables (refer to 4.4.2, "Querying Catalog Tables" on page 69). Some arguments of the catalog functions can use pattern-matching. In this example, `SQLColumns()` is using pattern-matching to specify the table schema and the table name.

Each pattern-matching argument can contain:

- The underscore "_" character which stands for any single character.
- The percent "%" character which stands for any sequence of zero or more characters.
- The pattern-matching argument is case sensitive.

```
...
SQLColumns(hstmt, NULL, 0, table_schem.s, SQL_NTS,      1
                table_name.s, SQL_NTS, (SQLCHAR *)"%", SQL_NTS);

SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER) table_schem.s, 129,
                &table_schem.ind);      2

SQLBindCol(hstmt, 3, SQL_C_CHAR, (SQLPOINTER) table_name.s, 129,
                &table_name.ind);

SQLBindCol(hstmt, 4, SQL_C_CHAR, (SQLPOINTER) column_name.s, 129,
                &column_name.ind);

SQLBindCol(hstmt, 6, SQL_C_CHAR, (SQLPOINTER) type_name.s, 129,
                &type_name.ind);

SQLBindCol(hstmt, 7, SQL_C_LONG, (SQLPOINTER) & length,
                sizeof(length), &length_ind);

while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {     3
    printf("%-14.14s %-8.8s %-17.17s %-18.18s %-6.6ld\n",
    table_name.s, table_schem.s, column_name.s, type_name.s, length);
}
...
```

Figure 53. Querying Using SQLColumns()

Figure 53 shows part of the listcol.c application that calls the SQLColumns() function **1**.

2 To be able to use the result set in application variables, we need to bind it. We call SQLBindCol() to bind the result set to the application variables. The second argument of SQLBindCol() specifies the number of the column in a sequential manner from left to right starting at 1. SQLBindCol() returns 17 columns in the result set (refer to the *Function* chapter in the *Call Level Interface Guide and Reference Book*).

3 Finally, we call SQLFetch() to retrieve any bound columns. The data transfer from the DBMS to the application occurs when we call this function. As the data is transfered, data conversions may also occur as defined in the third argument of SQLBindCol(). In this sample, the data type of the columns, Table Schema, Table Name, Column Name, and Type Name, will be converted to the SQL_C_CHAR data type. The data type of Length column will be converted to the SQL_C_LONG data type.

Running the listcol.c program listed in A.1, "CLI Example (listcol.c)" on page 143 will give the result shown in Figure 54 on page 88.

```
Enter Database Name : sample
Enter User Name     : db2
Enter Password for db2 : db2

--- Connected to database : sample ---

Enter Table Schema Name Search Pattern (in uppercase) :
DB%
Enter Table Name Search Pattern (in uppercase) :
EMPL%

TABLE          SCHEMA   COLUMN            TYPE                LENGTH
-------------- -------- ----------------- ------------------ ------
EMPLOYEE       DB2      EMPNO             CHARacter          000006
EMPLOYEE       DB2      FIRSTNME          VARCHAR            000012
EMPLOYEE       DB2      MIDINIT           CHARacter          000001
EMPLOYEE       DB2      LASTNAME          VARCHAR            000015
EMPLOYEE       DB2      WORKDEPT          CHARacter          000003
EMPLOYEE       DB2      PHONENO           CHARacter          000004
EMPLOYEE       DB2      HIREDATE          DATE               000010
EMPLOYEE       DB2      JOB               CHARacter          000008
EMPLOYEE       DB2      EDLEVEL           SMALLINT           000005
EMPLOYEE       DB2      SEX               CHARacter          000001
EMPLOYEE       DB2      BIRTHDATE         DATE               000010
EMPLOYEE       DB2      SALARY            DECimal            000009
EMPLOYEE       DB2      BONUS             DECimal            000009
EMPLOYEE       DB2      COMM              DECimal            000009
Disconnecting .....
```

Figure 54. The Result of Running listcol

4.5.2 Using CLOB Locator

The following example shows how to search the resume CLOB column in table EMP_RESUME from the sample database provided with DB2.

First, it presents a list of employee numbers and names. Then, given an employee number, the application will search the “Interests” section of the RESUME column by using a LOB locator. Only the result set will be sent from the database server to the application.

```
/* Include files, variables and SQL statements definition */
...
/* Initialization section */
...

/* Transaction Processing section */
   rc = SQLAllocStmt(hdbc, &hstmt);     1
   rc = SQLExecDirect(hstmt, stmt1, SQL_NTS);
   CHECK_STMT(hstmt, rc);

   rc = SQLBindCol(hstmt,1,SQL_C_CHAR,Empno.s,7,&Empno.ind);  2
   rc = SQLBindCol(hstmt, 2,SQL_C_CHAR, Name.s, 30,&Name.ind);
   CHECK_STMT(hstmt, rc);

   printf("\nEmpno   Name \n");                        3
   printf("------- --------------------\n");
   while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
        printf("%-6s %-30s \n", Empno.s, Name.s);
   }
   if (rc != SQL_NO_DATA_FOUND)
       check_error(henv, hdbc, hstmt, rc, __LINE__, __FILE__);

   rc = SQLFreeStmt(hstmt, SQL_CLOSE); CHECK_STMT(hstmt, rc);
   rc = SQLFreeStmt(hstmt, SQL_UNBIND); CHECK_STMT(hstmt, rc);
   rc = SQLFreeStmt(hstmt, SQL_RESET_PARAMS); CHECK_STMT(hstmt, rc); .
   rc = SQLSetParam(hstmt, 1, SQL_C_CHAR, SQL_CHAR, 7,     4
                    0, Empno.s, &Empno.ind);
   CHECK_STMT(hstmt, rc);

   printf("\n>Enter an employee number:\n");
   gets((char *)Empno.s);

   rc = SQLExecDirect(hstmt, stmt2, SQL_NTS);       5
   CHECK_STMT(hstmt, rc);
   rc = SQLBindCol(hstmt, 1, SQL_C_CLOB_LOCATOR, &ClobLoc1, 0,
                   &pcbValue);
   rc = SQLFetch(hstmt);
   rc = SQLAllocStmt(hdbc, &lhstmt);          6

   rc = SQLGetLength(lhstmt, SQL_C_CLOB_LOCATOR, ClobLoc1,    7
                     &SLength, &Ind);
   rc = SQLGetPosition(lhstmt,SQL_C_CLOB_LOCATOR, ClobLoc1, 0,  8
              (SQLCHAR *)"Interests", 9, 1, &Pos1, &Ind);
   CHECK_STMT(lhstmt, rc);

   rc = SQLFreeStmt(lhstmt, SQL_CLOSE); CHECK_STMT(lhstmt, rc);
   buffer = (SQLCHAR *)malloc(SLength - Pos1 + 1);
   rc = SQLGetSubString(lhstmt,SQL_C_CLOB_LOCATOR,ClobLoc1,Pos1,  9
        SLength - Pos1, SQL_C_CHAR, buffer, SLength - Pos1 +1,
        &OutLength, &Ind);
   CHECK_STMT(lhstmt, rc);

   printf("\nEmployee #: %s\n %s\n", Empno.s, buffer);

/* Terminate section */
...
```

Figure 55. Example Using CLOB Locator: lookres.c

The Transaction Processing section begins allocating the statement handle **1**. Then, it will execute the SQL statement.

In order to use the arrays for retrieving data, you must bind them to their respective columns **2**.

3 will display the result of the select statement (employee number and name) stored in the arrays.

The SQLFreeStmt() statements will close the cursor and then bind and reset the parameters for the first SQL statement executed in order to use the same statement handle for the second statement.

4 assigns storage for a new parameter to get the employee number, which is later prompted for.

5 executes the second SQL statement. To retrieve its result, the SQLBindCol() function assigns storage for the LOB column locator and also specifies its type. Then, the result is retrieved by a fetch function.

6 allocates a new statement handle for working with the LOB column, while **7** gets the total length of the LOB column and **8** gets the starting position for the specific text, the “Interests” section for this example.

The next step is to close the cursor and define a new buffer size to contain the LOB portion selected.

9 will search the CLOB locator to find “Interests” and get the substring of resume from this position to the end. Then, it will print the “Interests” section of the employee’s resume.

The SQLFreeStmt() statements will free the two statement handles allocated in the application and its associated resources. Finally, it will commit the transaction and end with the Termination section.

Run the lookres.c program listed in A.2, “CLI LOB Example (lookres.c)” on page 145 using sample database as a Server name, and any user ID and password with SELECT privilege on this database. It will show the following:

```
$ lookres
>Enter Server Name:
sample
>Enter User Name:
db2
>Enter Password:
db2
>Connected to sample

Empno   Name
------- --------------------
000130 DOLORES  QUINTANA
000140 HEATHER  NICHOLLS
000150 BRUCE    ADAMSON
000190 JAMES    WALKER

>Enter an employee number:
000190

Employee #: 000190
 Interests

  o   Wine tasting
  o   Skiing
  o   Swimming
  o   Dancing
>Disconnecting .....
$
```

Figure 56. Running the lookres.c Sample Program

Chapter 5. Open Database Connectivity (ODBC)

Microsoft's Open Database Connectivity (ODBC) is found on many platforms. It provides a standard set of routines that can be used with different types of database systems. This chapter discusses how ODBC is implemented under DB2 for AIX and what needs to be done to enable an ODBC application to access the DB2 databases.

5.1 Overview

The ODBC is a callable SQL interface that has been defined by Microsoft. The ODBC interface defines the following:

- A set of function calls that implement SQL statements
- The SQL syntax which is based on the X/Open and SAG SQL CAE draft specification (1991)
- A set of standard error codes
- A standard way to connect and log on to a database management system
- A set of data-type representations

When running in an ODBC environment, there are a number of parts that make up the connection. These are:

1. The Application
2. The ODBC Driver Manager
3. The Driver
4. The Data Source

Figure 57 shows the layers or parts in the ODBC environment. You can see under each layer the owner or application that provides the layer.

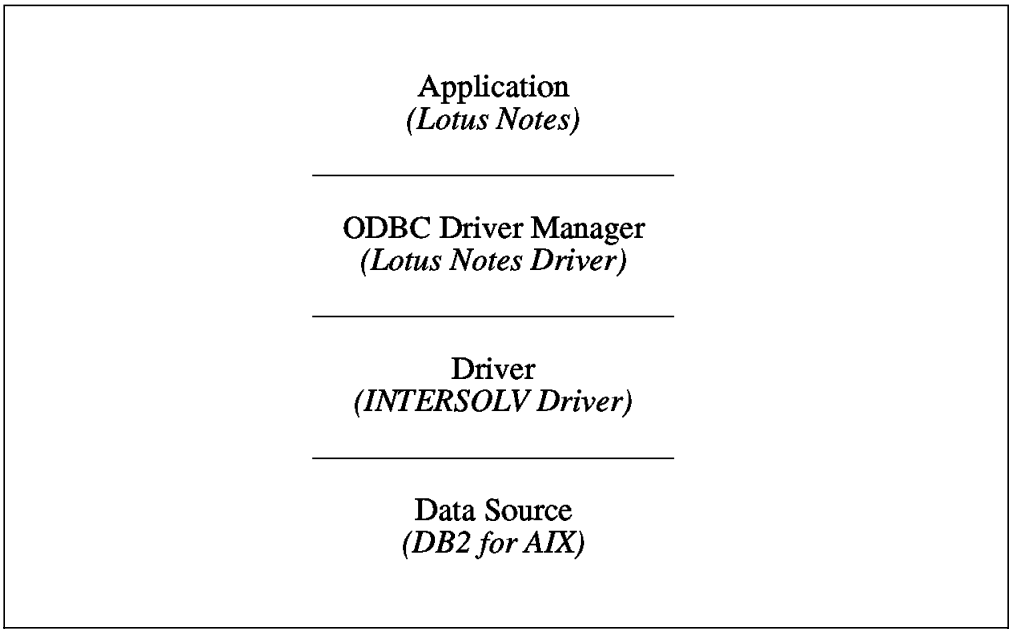


Figure 57. ODBC Environment

The Application

The application invokes the ODBC function calls to submit an SQL request, then retrieveS the results. In order to do this, the application must first request a connection to the database. Once a connection has been established, the next step is to send the SQL request. When the request has been processed by the database manager, the results can be requested by the application. The application then can pass the results back to the user; or, if an error occurred, it should process the error condition.

The ODBC Driver Manager

The Driver Manager provides the interface to the application. The functions of the Driver Manager include:

- Loading and unloading the database driver
- Performing status checking
- Managing multiple connections between applications and databases
- Map database names to a specific driver using the configuration file, odbc.ini

The Driver

The driver implements the ODBC function calls and interacts with the particular database. The driver will establish the connection to the database and then submit the SQL requests over the established connection. The driver also performs any necessary data conversions and maps any messages that may be returned from the database.

The Data Source

The data source is a database management system. The data source may be a text file located in a directory, or it may be a relational database management system, such as DB2, located on a remote server.

As long as the appropriate driver can be loaded, the data source should not matter to the ODBC application being executed.

5.2 Configuring ODBC

The configuration of your ODBC environment may depend on the platform that you are using. This section will look at configuring the AIX client and OS/2 client environments.

5.2.1 Configuring ODBC on OS/2

DB2 Client Application Enabler for OS/2 provides the necessary utilities and drivers for accessing DB2 databases via ODBC. These databases may be local if you are running the DB2 Server or DB2 Single-User on the OS/2 machines, or they may be cataloged, remote databases.

In this example, we will assume that the databases are on a remote DB2 for AIX Server.

1. Install Client Application Enabler Code

The Client Application Enabler code is installed using the `install` program. When the installation is complete, you should find the IBM DATABASE 2 folder located on you OS/2 desktop. A sample of the contents of this folder are shown in Figure 58 on page 95.

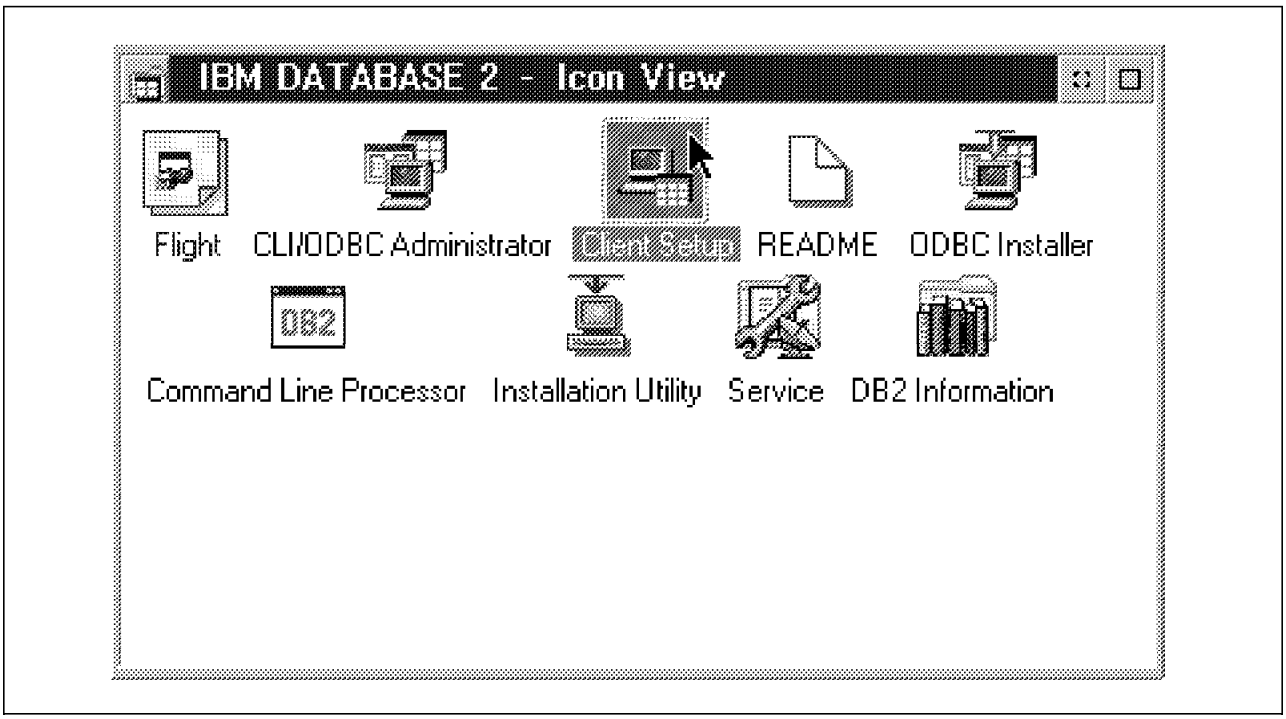


Figure 58. Sample CAE/2 Folder

2. Configure the Client

   From the DB2 folder, you can configure the CAE/2 client. By selecting the **Client Setup** application, you will be presented with the screen shown in Figure 59 on page 96.

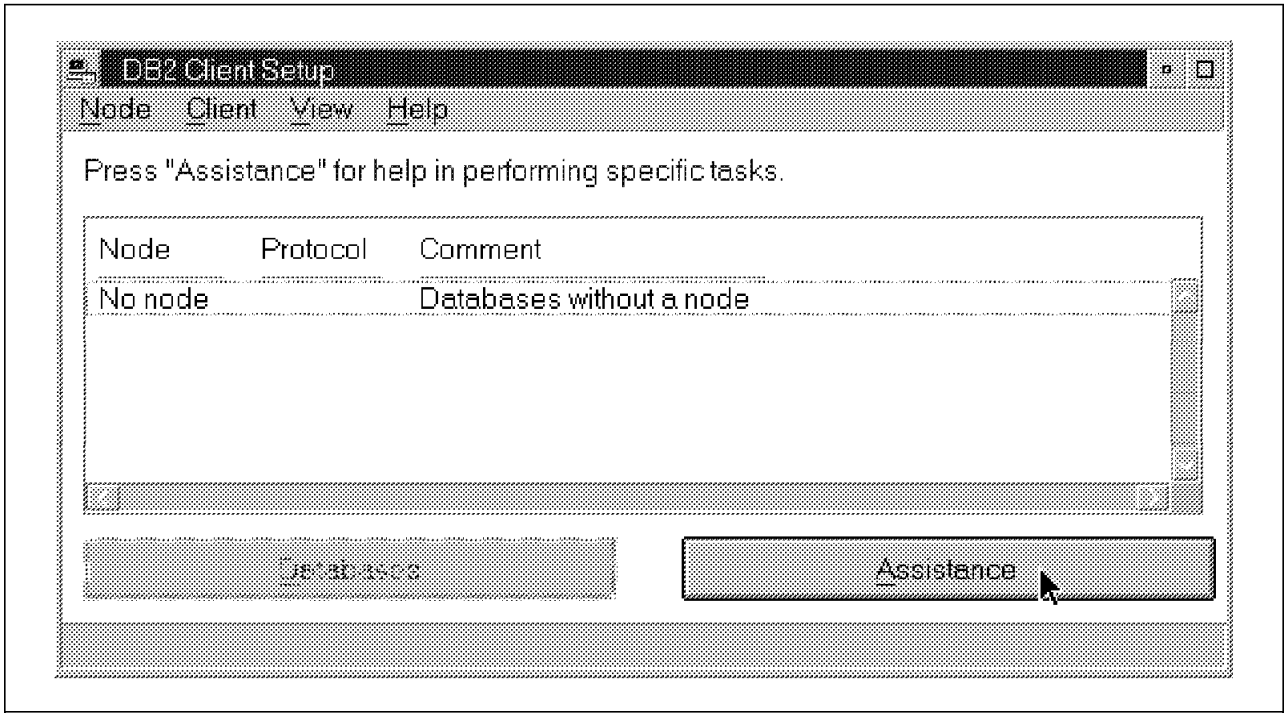


Figure 59. CAE/2 Client Setup

From this screen, you may either select the Assistance button, or use the menu options listed across the top of the screen. The basic tasks that must be performed, using either method, is to catalog a node and database.

3. Catalog the Remote Node

   When you select to add a new node through the client setup application, you will be presented with a screen similar to the one shown in Figure 60 on page 97. In this example, we are adding a node called GUNDAGAI, using the TCP/IP protocol and the TCP service name db2. This service must be added to the TCP/IP services file. You should refer to the DB2 installation documentation for further details on configuring a TCP/IP service.

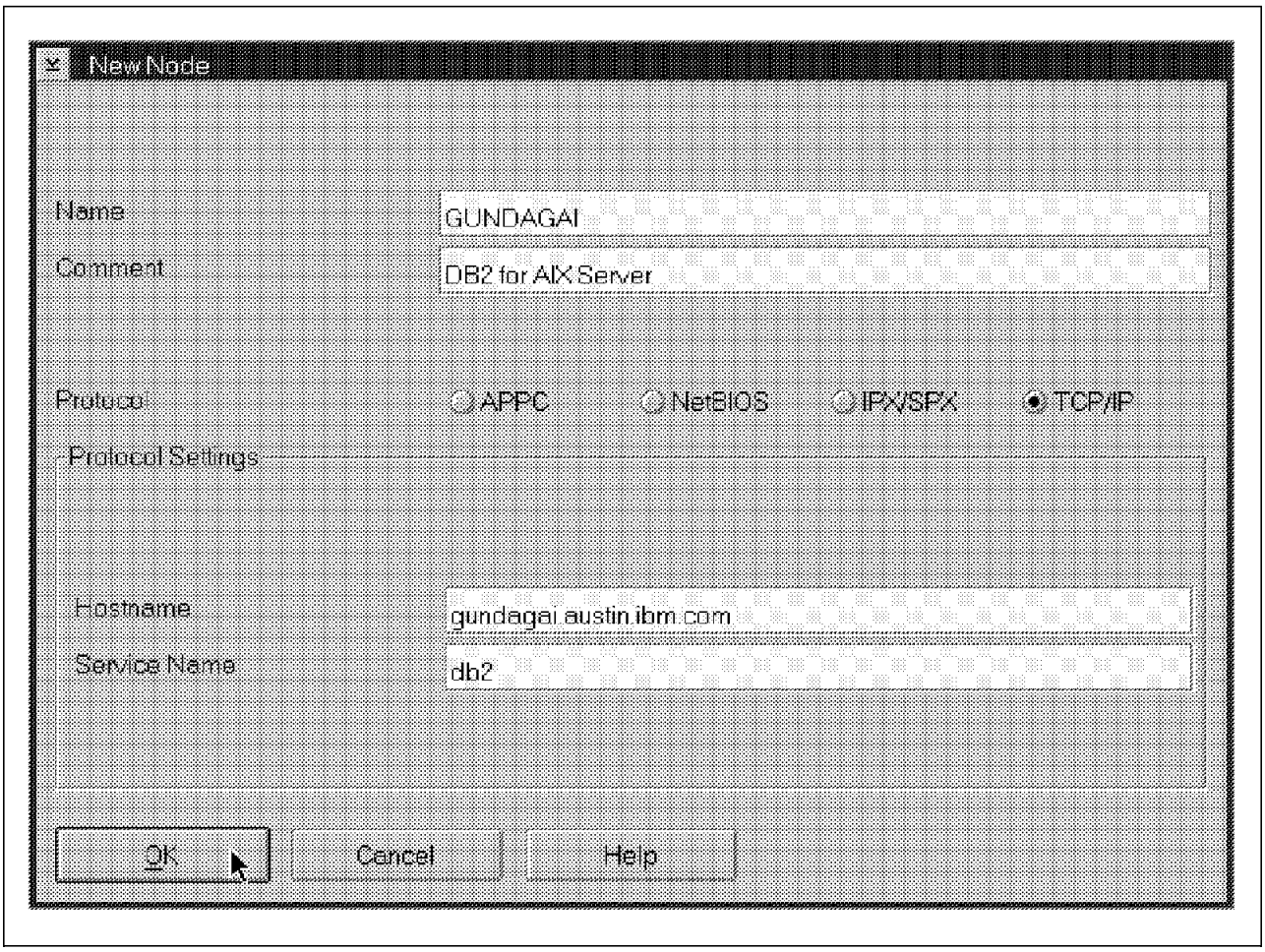


Figure 60. Cataloging a Remote Node

4. Catalog the Remote Database

   Once the node has been cataloged, you will be able to catalog the database. Figure 61 on page 98 is an example of cataloging the SAMPLE database, which is located on the server GUNDAGAI.

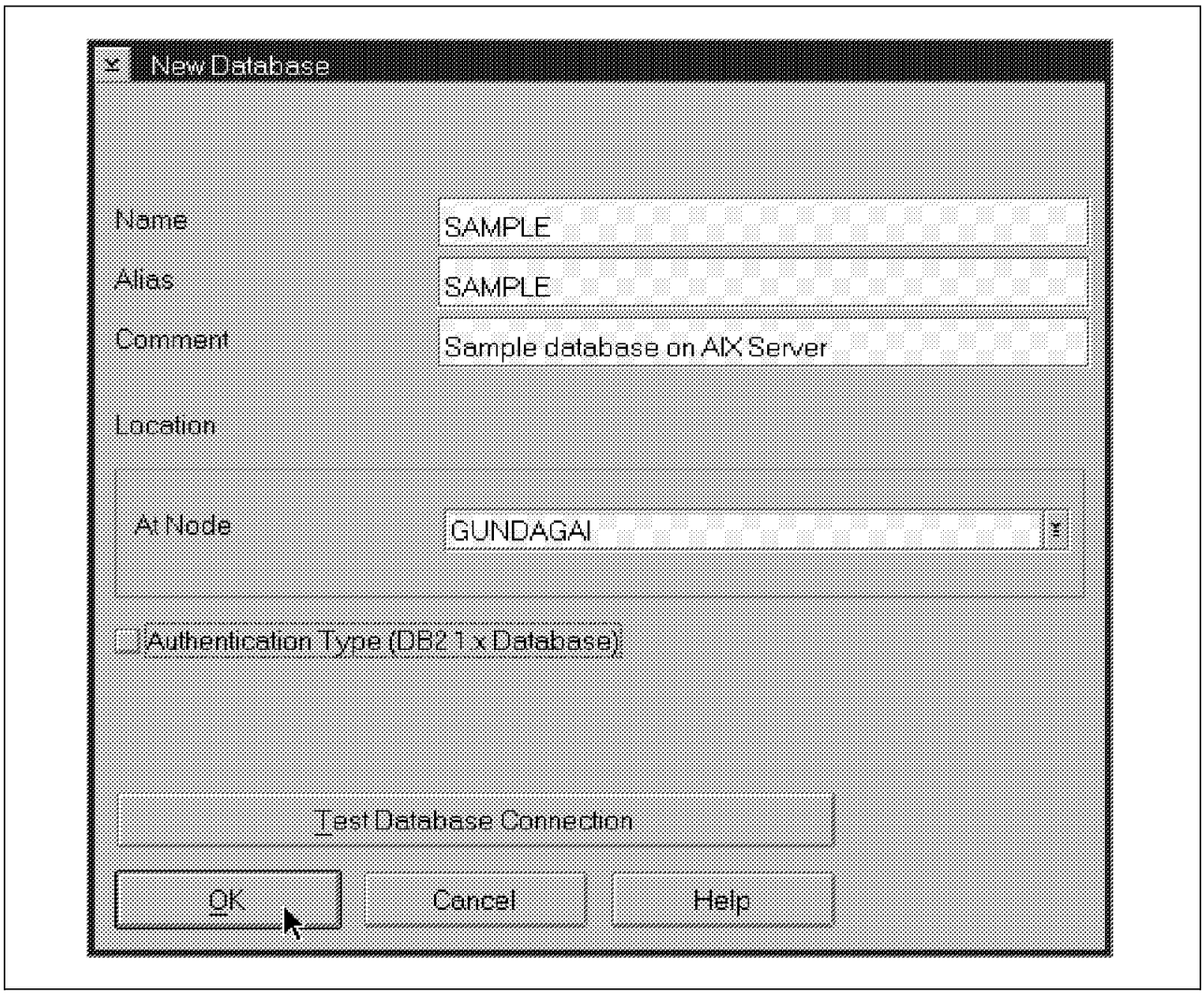


Figure 61. Cataloging a Remote Database

It is wise to test the database connection using the Test Database Connection button on the window. Alternatively, you may test the database connection at a later date by using the Command Line Processor and the command, `connect to database sample`.

To connect to a DB2 server which uses server authentication, you will need to know a user ID on the server machine and the corresponding password. Connections from an OS/2 client capitalize the password before it is passed to the server. Because of this, you need to make sure the AIX user ID has no lowercase letters in the password.

Once you are able to successfully connect to the database, you can proceed with defining the database as an ODBC data source.

5. Run the ODBC Installer

   The first step in using ODBC with DB2 is to install the ODBC drivers. This can be done by executing the ODBC Installer program located in the IBM DATABASE 2 folder. This will place an ODBC folder on your OS/2 desktop.

6. Configure the ODBC Data Source and Driver

   From the ODBC folder on the OS/2 desktop, you can start the ODBC Administrator. This application will present you with a screen similar to the

one shown in Figure 62 on page 99. This screen lists all the ODBC data sources that are available on your workstation.

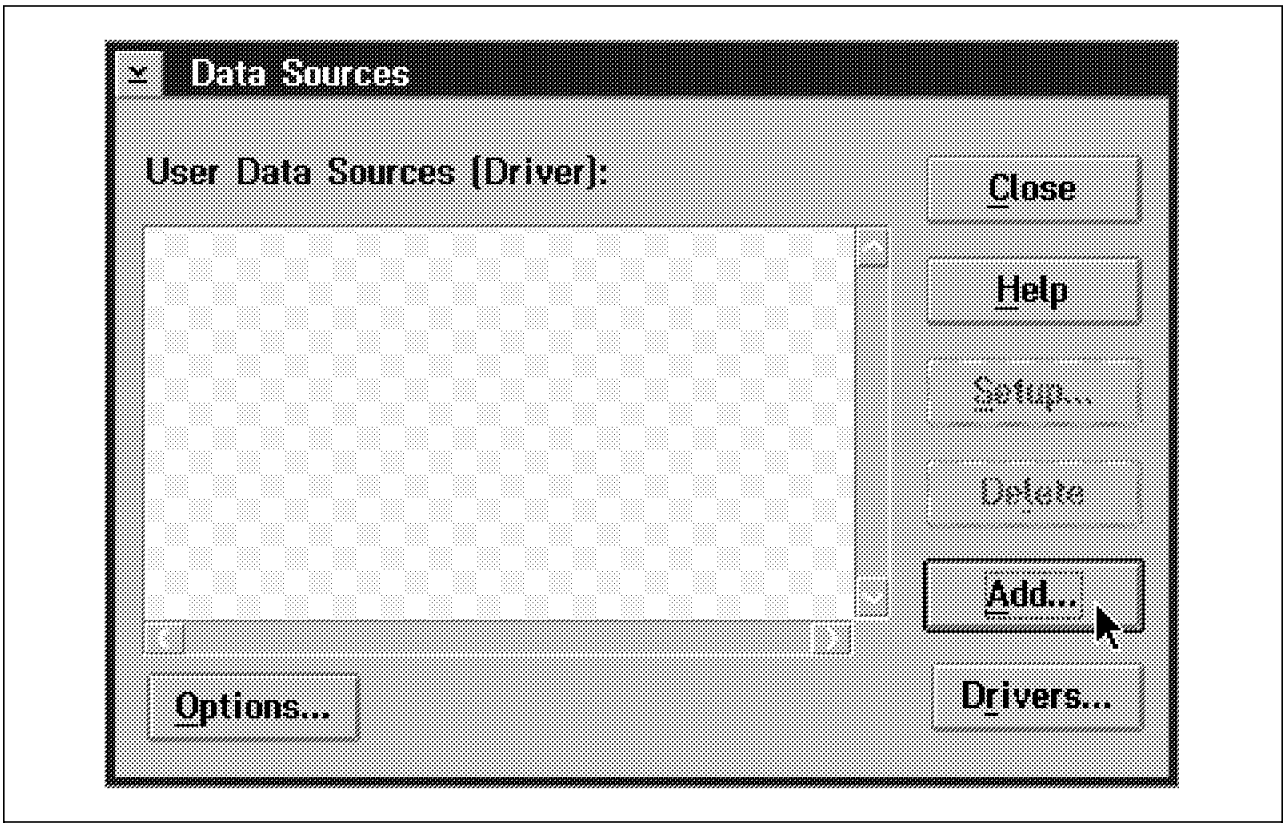


Figure 62. ODBC Administrator - Data Sources

To define the DB2 sample database as a data source, you should select the **Add**, button. This will present you with a list of the known ODBC drivers that are available= as shown in Figure 63 on page 100.

You should select the **DB2 ODBC driver**, and click **OK** to continue.

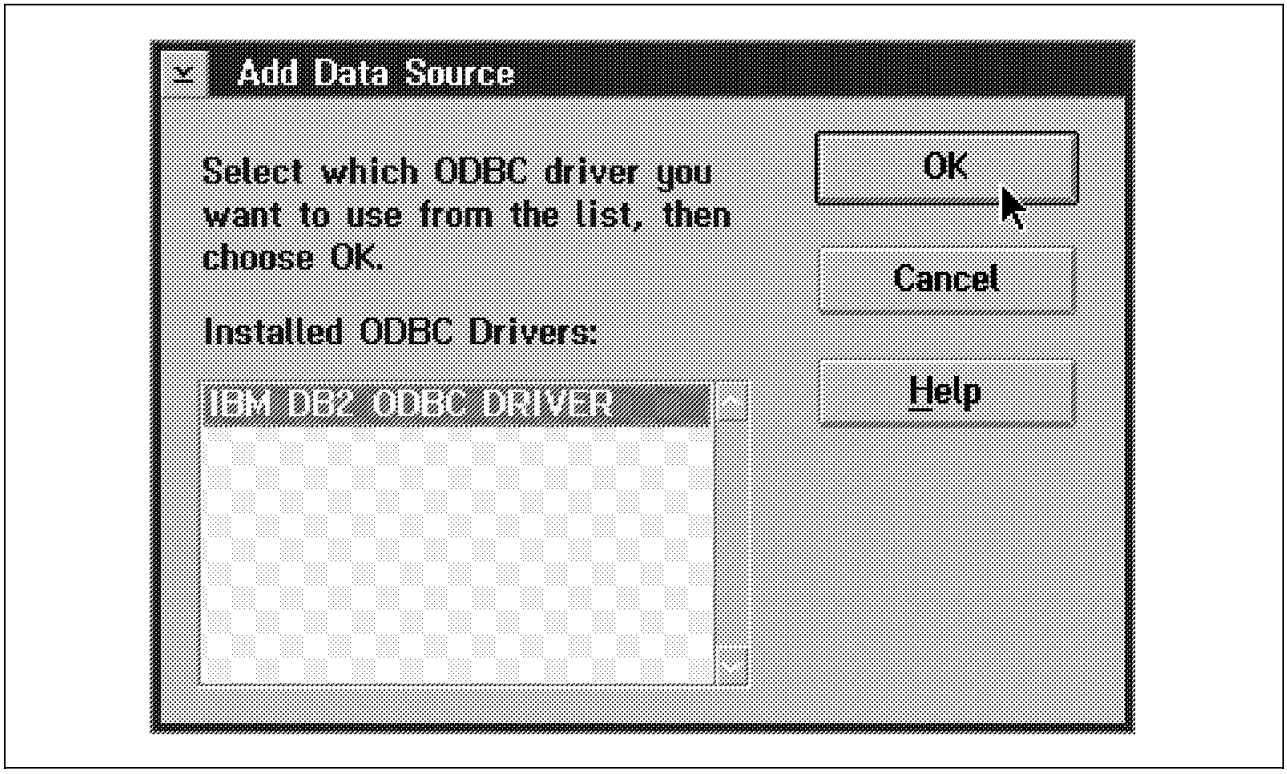


Figure 63. Available ODBC Drivers

The final step is to select the database you wish to make available as your data source. The cataloged DB2 database will be listed for you to select from.

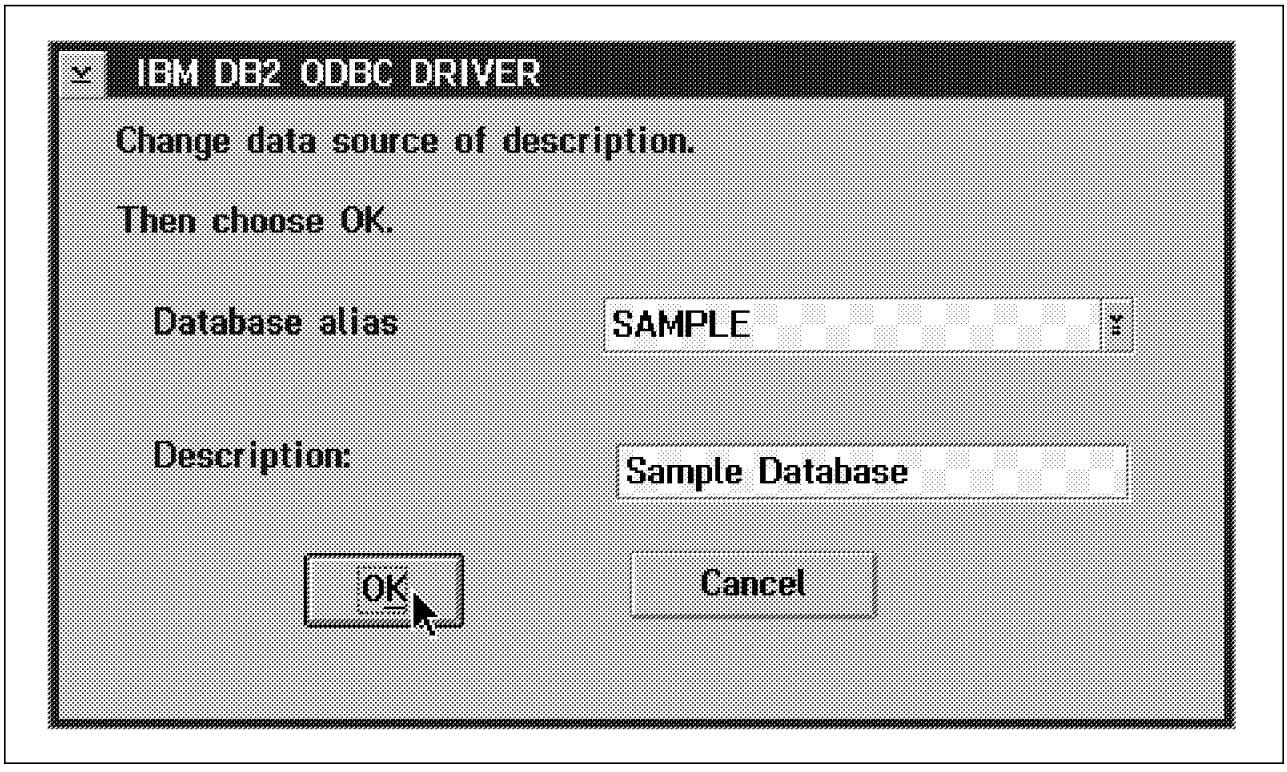


Figure 64. ODBC Data Source and Description

Once you have selected the data source, your ODBC application should be able to connect to the database.

In the IBM DATABASE 2 folder on your OS/2 desktop, you also will find a CLI/ODBC Administrator application. Either this application or the ODBC Administrator may be used to configure your data sources and set the different options available.

Once these steps have been completed, you should be ready to access your DB2 database via an ODBC application.

5.2.2 Configuring ODBC on AIX

ODBC applications have not existed on UNIX platforms for as long as they have existed on the Intel platform. Because of this, some of graphical configuration tools are not yet available, and the location of the drivers may vary depending on the supplier of the ODBC drivers being used.

DB2 for AIX provides three different methods for accessing DB2 databases through the ODBC interface. These methods are:

- Call Level Interface

  The Call Level Interface provides support for Level 1 and most Level 2 ODBC calls.

- DB2 ODBC Drivers

  DB2 supplies an ODBC Driver Manager and driver that may be used by your ODBC applications.

- Other ODBC Drivers

  You may use a third-party ODBC Driver Manager and driver for access to DB2 databases.

Regardless of which driver you choose to install, you still will need to configure the DB2 Client Application Enabler for AIX so that you are able to access the server databases.

The Client Application Enabler software is installed in the same manner as other AIX products, that is, by using the `installp` command or the `smit` command.

Once the Client Application Enabler software has been installed, you will need to perform the following steps:

Log in as the root user to perform the following steps:

1. Configure a DB2 instance on the client.

   Before you configure an instance, you need to create or select a user ID that will become the instance owner. For this example, we will use the user ID db2.

   To create the instance, you need to issue the command:

   ```
   /usr/lpp/db2_02_01/instance/db2icrt db2
   ```

   The user db2 will become the instance owner, and db2′s primary group will become the system administration group for that instance.

2. Configure the TCP/IP service being used.

You will need to determine the service being used by the server machine that you wish to connect to. This can be found by running the db2 command, `get database manager configuration` and checking the value of the "Service Name" variable. You should then find this name in the server's /etc/services file. Note down the entry and create an identical entry in the client's /etc/services file.

You should note that if Version 1 clients are being used, there will be two entries in the /etc/services file. Refer to the installation instructions supplied with DB2 for further information.

Log in as the instance owner to perform the following steps:

3. Set up the user's environment.

   You will need to set up the instance owner's environment. The process to do this will depend on the shell you are running under.

   - For Korn (ksh) or Bourne Shell (bsh, sh)

     You will need to include the environment setting from the file $HOME/sqllib/db2profile. This can be done by executing the following command or by including it in the file $HOME/.profile:

     ```
     . $HOME/sqllib/db2profile
     ```

   - For C Shell (csh)

     You will need to include the environment setting from the file $HOME/sqllib/db2cshrc. This can be done by executing the following command or by including it in the file $HOME/.login:

     ```
     source $HOME/sqllib/db2cshrc
     ```

4. Catalog the server node.

   Once TCP/IP communications has been established, you may catalog the server node. This can be done by using the following DB2 command:

   ```
   $ db2 catalog tcpip node gundagai \
     remote gundagai.austin.ibm.com server db2
   ```

   You can check that the node was cataloged correctly by using the DB2 command:

   ```
   $ db2 list node directory
   ```

5. Catalog the database.

   Once the server node is cataloged, you can then catalog the database. To catalog the sample database, you can use the following command:

   ```
   $ db2 catalog database sample as sample at node gundagai
   ```

   You can check the cataloged database with the command:

   ```
   $ db2 list database directory
   ```

6. Test the connection to the remote database.

   To test the connection, you can try connecting to the sample database. To connect to a database on a server node, using server authentication, you will need a valid user ID and password on the server.

   To connect to the database, you can use the command:

   ```
   $ db2 connect to sample user myuserid
   ```

   You will be prompted for a password, and then a connection message should be returned.

Now that the DB2 client is able to connect to databases located on the server node, the next step is to configure the ODBC environment. The ODBC configuration used will depend on the ODBC drivers that you are using. The following sections are guidelines to the setup that may be required.

5.2.2.1 ODBC via the Call Level Interface

For many database connections, the application will establish the environment and connection settings. It is possible to configure many of the options for a CLI connection using the db2cli.ini. configuration file. This file is in the directory sqllib/cfg, which will be found in the DB2 instance owner's home directory.

The options that may be set here are discussed in the *IBM DATABASE 2 Call Level Interface Guide and Reference - for common servers*. The options set here may be overridden by the application and are not required for your ODBC connection to DB2 databases.

Before using the CLI interface, you need to bind the CLI components/packages to the database, if this has not already been done. The binding process is discussed in 4.3, "CLI Application Configuration and Execution" on page 52. This will need to be done once for each database that will be accessed.

Once you have bound the required packages to the database and, optionally, set up the db2cli.ini file, you are ready to develop ODBC applications that will access DB2 data sources via the CLI/ODBC interface.

5.2.2.2 ODBC via the DB2 ODBC Drivers

If you have applications that access multiple data sources, you may prefer to define the ODBC drivers in the odbc.ini file. The odbc.ini file is used by DB2 to determine the ODBC driver being used for a particular data source. This file will need to exist in the user's home directory and be prefixed with a dot (.). You should also have a file called odbcinst.ini. This also needs to be located in the user's home directory and prefixed with a dot.

The odbcinst.ini file is used by the ODBC Driver Manager to determine which drivers are available. The example shown in Figure 65 lists the IBM DB2 ODBC Driver installed and shows where it is located.

```
[ODBC Drivers]
IBM DB2 ODBC DRIVER=Installed

[IBM DB2 ODBC DRIVER]
Driver=/home/db2/sqllib/lib/db2.o
```

Figure 65. Sample odbcinst.ini Configuration File

Figure 66 on page 104 is an example of the odbc.ini file. This example lists the sample database as a data source.

1 is the list of available ODBC data sources.

2 is the stanza for the data source SAMPLE, and it provides the location of the driver **3** and a description of the data source **4**. Other ODBC options for this data source may be defined in this stanza.

```
[ODBC Data Sources]                     1
SAMPLE=IBM DB2 ODBC DRIVER

[SAMPLE]                                2
Driver=/home/db2/sqllib/lib/db2.o       3
Description=Sample DB2 ODBC Database    4
```

Figure 66. Sample odbc.ini Configuration File

5.2.2.3 ODBC via Third-Party Drivers

Configuration of other ODBC drivers may vary depending on the supplier of the code. In general, the configuration of the driver after it is installed will be similar to that of the DB2 ODBC Driver.

You will need to list the driver in the odbcinst.ini configuration file, and the data sources will need to be listed in the odbc.ini file.

Figure 67 shows an example of the odbcinst.ini configuration file with both the DB2 ODBC Driver and the INTERSOLV ODBC Driver for DB2.

```
[ODBC Drivers]
IBM DB2 ODBC DRIVER=Installed
INTERSOLV 2.10 DB2=Installed

[IBM DB2 ODBC DRIVER]
Driver=/home/db2/sqllib/lib/db2.o

[INTERSOLV 2.10 DB2]
Driver=/opt/odbc/dlls/qedb208.so
Setup=/opt/odbc/dlls/qedb208.so
APILevel=1
ConnectFunctions=YYN
DriverODBCVer=02.10
FileUsage=0
SQLLevel=1
smProcessPerConnect=Y
```

Figure 67. Sample odbcinst.ini Configuration File with INTERSOLV Drivers

A sample odbcinst.ini is supplied with the INTERSOLV DataDirect ODBC Pack. You also will find an example of the odbc.ini file in Figure 68 on page 105.

```
[ODBC Data Sources]
SAMPLE=IBM DB2 ODBC DRIVER
qedb2=INTERSOLV DB2 ODBC Driver

[SAMPLE]
Driver=/home/db2/sqllib/lib/db2.o
Description=Sample DB2 ODBC Database

[qedb2]
Driver=/opt/odbc/dlls/qedb208.so
Description=INTERSOLV DB2 ODBC Driver
ServerName=
LogonID=
```

Figure 68. Sample odbc.ini Configuration File with INTERSOLV Drivers

5.3 Programming with ODBC

Once your ODBC environment has been established, writing an application using the ODBC library of call is very similar to using the CLI calls. You should read Chapter 4, “Call Level Interface” on page 39 if you are unfamiliar with programming applications that use an ODBC/CLI interface.

5.3.1 Compiling and Linking Applications

The compilation and linking of your ODBC applications will depend upon the ODBC driver that you are using.

Using DB2 Call Level Interface

The compilation and linking of applications that use the CLI library of calls is discussed in 4.3.3, “ Compiling and Linking Applications” on page 59.

Using ODBC Libraries

DB2 supplies an ODBC library as discussed earlier. The ODBC libraries are located in the directory sqllib/odbclib/lib, which is under the instance owner′s home directory.

The library libodbcinst.a is used by the Driver Manager, and the library libodbc.a supplies the routines for the ODBC application. If you are using the ODBC libraries from another source, then you should read the configuration documentation supplied with the ODBC libraries.

User Environment

You may find that you need to configure some environment variables before your applications will work in the ODBC environment. The value of these variableS might vary because of the flexibility of the UNIX environment. The variables common to most environments are:

LIBPATH This environment variable specifies the directories that will be searched for libraries. On most UNIX systems, if this variable is not set, the directories /lib and /usr/lib will be searched.

PATH This environment variable specifies the directories that will be searched for executables.

When you are using the Call Level Interface, the libraries will have symbolic links created in the standard UNIX directories; so you will not be required to modify environment variables. If you find that these links do not exist, you will need to run the command /usr/lpp/db2_02_01/cfg/db2ln as the root user.

5.4 Example ODBC Application Environment

The following environments are an example that may be used to to compile and execute an ODBC application.

5.4.1 Compiling an ODBC Application

To compile an ODBC application, you need to have your environment set so that all the required libraries and header files can be found by the compiler being used. The following examples can be used as a guideline to the configurations required.

DB2 Call Level Interface

If you have created the appropriate library and included file links as previously discussed, then there are no requirements with DB2 for additional directories to be included in you compilation environment. However, to illustrate the libraries and include files being used, these examples will assume the links do not exist.

```
# Set to DB2 Instance Owner's Home Directory
DB2=/home/db2

# Define the Compiler, Loader, Flags and Libraries
CC=xlC
LD=xlC
CCFLAGS=-I$(DB2)/sqllib/include
LDFLAGS=-L$(DB2)/sqllib/lib
CLIBS=-ldb2

# Target Lines
listcol: listcol.o
        $(LD) -o $@ $(LDFLAGS) listcol.o $(CLIBS)
```

Figure 69. Sample DB2 CLI Makefile

Figure 69 is an example of a makefile that could be used to compile an ODBC application using the DB2 CLI library. The environment variables that need to be defined are all included in the file, sqllib/db2profile. This file is located in the instance owner's home directory.

No other configuration or setup is required if you are using the CLI functions.

ODBC Libraries

If you decide to use the DB2 ODBC libraries, then the makefile you would use may look like the one shown in Figure 70 on page 107.

You also may choose to use another vendor's ODBC libraries. In this environment, you would need to be aware of where the software is installed. Figure 71 on page 107 is an example using the INTERSOLV DataDirect ODBC Pack. The software is installed in the directory /opt/odbc.

```
# Set to DB2 Instance Owner's Home Directory
DB2=/home/db2

# Define the Compiler, Loader, Flags and Libraries
CC=xlC
LD=xlC
CCFLAGS=-I$(DB2)/sqllib/include
LDFLAGS=-L$(DB2)/sqllib/odbclib/lib
CLIBS=-lodbc

# Target Lines
listcol: listcol.o
        $(LD) -o $@ $(LDFLAGS) listcol.o $(CLIBS)
```

Figure 70. Sample DB2 ODBC Makefile

When using the ODBC libraries, it also may be necessary to include the directory containing the ODBC libraries in the LIBPATH environment variable. For example, using the INTERSOLV drivers, you may need to set up the LIBPATH environment as follows:

```
$ LIBPATH=/lib:/usr/lib:/opt/odbc/dlls
$ export LIBPATH
```

```
# Define the Compiler, Loader, Flags and Libraries
CC=xlC
LD=xlC
CCFLAGS=-I/opt/odbc/include
LDFLAGS=-L/opt/odbc/dlls
CLIBS=-lodbc

# Target Lines
listcol: listcol.o
        $(LD) -o $@ $(LDFLAGS) listcol.o $(CLIBS)
```

Figure 71. Sample ODBC Makefile

Chapter 6. Database Extenders

As a powerful database, IBM DATABASE 2 Version 2 stores and manages not only traditional numeric and character data but also complex data such as Character Large Objects (CLOB), Binary Large Objects (BLOB) and Double-Byte Character Large Objects (DBCLOB). To exploit these new complex data types in your applications, IBM introduces the DB2 Relational Extenders.

This chapter gives you a brief, general description of what the Industry Relational Extenders are by presenting the DB2 Text Extender as the first member of the DB2 Relational Extenders. Other extenders that include image, audio, video, and fingerprint capability will be made available.

6.1 Overview

With the emergence of new complex data types, such as CLOBs, BLOBs, and DCBLOBs, it becomes necessary to create applications that can use and manage these new data types. These complex data types must be able to be searched, accessed and manipulated through the standard SQL statements from within your applications. For example, an application searches a video clip based on its attributes, such as the length, the frame rate, the number of the frames, and so on.

The DB2 Relational Extenders are a set of IBM products that help users of DB2 Version 2 to exploit the large object (LOB) data types. The DB2 Relational Extenders make use of some of the supported features in DB2 Version 2 in the following ways:

- The DB2 Relational Extenders exploit DB2 Version 2′s support for large objects.
- The DB2 Relational Extenders define new data types and functions for image, audio, video, and text objects using the DB2 Version 2′s built-in support for User-Defined Types (UDT) and User-Defined Functions (UDF).
- The DB2 Relational Extenders use DB2 Version 2′s triggers to provide integrity checking across database tables to ensure the referential integrity of multimedia data.
- The DB2 Relational Extenders exploit the client/server model of DB2.

The DB2 Relational Extenders can help users by:

- Improving application development productivity and reducing the complexity of the application since users do not need to define data types and functions in the application. Data types and functions for LOBs are already defined by the extenders.
- Ensuring data consistency because users will use the same set of UDTs and UDFs defined by the extenders.
- Creating queries for any data types. Users can access LOBs data and traditional data together by writing only one SQL statement. That is because UDFs defined by the DB2 Relational Extenders can be invoked in the same way as other SQL functions.
- Managing the security, integrity and recovery protection of the LOBs since the objects processed by the extenders are also stored in DB2 databases.

6.1.1 The DB2 Relational Extenders Family

As a general definition, extenders can be classified into four categories:

1. Cross-industry, multimedia extenders
2. Cross-industry, non-multimedia extenders
3. Industry specific, multimedia extenders
4. Industry specific, non-multimedia extenders

Some examples of extenders in each category are shown in Table 20.

Table 20. Four Categories of Extenders

	Cross-Industry	**Industry Specific**
Multimedia	Text Audio Image Video	Fingerprint Medical ...
Non-multimedia	Scientific Functions Conversion Functions ...	Financial ...

The difference between:

- Cross-industry and Industry specific extenders is that Cross-industry extenders provide data types that are applicable to many different industries, while Industry specific extenders provide complex data types that have unique characteristics to match the requirements of a specific industries.
- Multimedia and non-multimedia differ in the data types that are supported. The multimedia provides the storage and processing of multimedia data types, while the non-multimedia extenders support more traditional data types.

For example, a fingerprint might have the same format for its storage as a static image of multimedia data. But, the attributes of a fingerprint are specific and may reflect an important piece of information in law enforcement applications. The Fingerprint Extender will provide a search function that can give the maximum performance and selectivity while identifying a fingerprint from a large number of similar prints.

IBM provides the following Cross-Industry Extenders:

- DB2 Text Extender
- DB2 Audio Extender
- DB2 Image Extender
- DB2 Video Extender

And supports the following Industry Specific Extender:

- The DB2 Fingerprint Extender

6.1.2 The Extender, Database and Application Relationship

As mentioned in 6.1.1, “The DB2 Relational Extenders Family” on page 110, there are several types of extenders. Each extender product has its own set of UDTs, UDFs and triggers which are specific for its purpose. For example, the DB2 Text Extender has a specific facility to do efficient text searching on a text document, while the DB2 Video Extender has a distinct facility to do the same efficient searching but based on the color contained in the specific time frame or some other video attributes.

Figure 72 shows the relationship between the Relational Extenders, the database and the application in a client/server model.

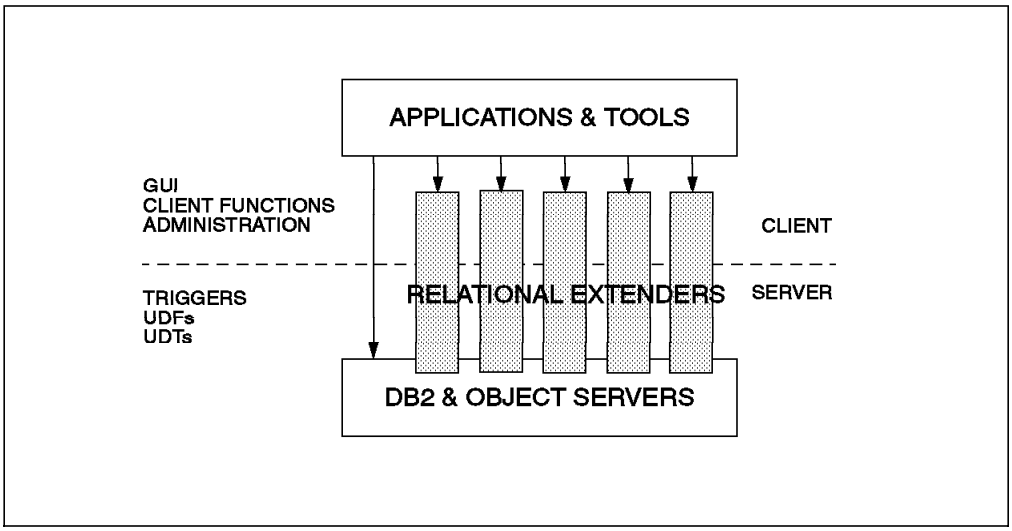


Figure 72. The Relationship between the Extender, the Database and the Application

There can be more than one extender product installed on a server and in use by an application. The application program uses the appropriate UDFs provided by a particular extender for a specific data type corresponding to the extender product. The application uses these UDFs in its SQL statements for processing the data on a table column.

As shown in Figure 72, an extender product consists of the following items:

1. User-Defined Types (UDT)

   The extender provides some UDTs that will be used by the extender’s Use-Defined Functions.

2. User Define Functions (UDF)

   These are the new SQL functions provided by the extenders that you can use for searching data.

3. Triggers

   Triggers are used to maintain the internal structure or indexes of a complex data type, such as maintaining a log of changes to the tables or a periodic update of the index tables.

4. Administration

   The administration component provides the commands to perform administrative operations associated with each extender. For example, establishing and maintaining indexing or pre-indexing data.

The extenders usually provide a command line interpreter (similar to the Command Line Processor of DB2) to submit the administration commands.

5. Client Functions

   These are Application Programming Interface (API) functions used at the extender client. These functions are written in a programming language (for example, C or C++) according to the client platform. These functions can be called in C programs to perform useful manipulation operations on their data once it has been transferred to the client application.

6. Graphical User Interface (GUI)

   For some extenders, a Graphical User Interface has been included. This helps to simplify the administration tasks.

6.2 DB2 Text Extender

In this chapter, we discuss the DB2 Text Extender, which is the first member of the DB2 Relational Extenders family.

The DB2 Text Extender extends the work of SQL statements to search for information on unstructured text stored in a DB2 database.

6.2.1 Advantages of the DB2 Text Extender

The DB2 Text Extender allows application programs to have full-text retrieval capabilities in their SQL queries for text documents. Using the DB2 Text Extender, an application can perform the following tasks.

- Search through many large text documents.
- Access any kind of text documents, including word-processing documents in their original native form.
- Search for documents that contain specific text that can be defined using one of the following search methods:
  - **Synonym** search, where the result may be documents which contain not only the exact word specified but also synonyms of this word.
  - **Proximity** search, where the result are documents that have words which are in close proximity to each other or are within a sentence of each other.
  - **Boolean** search, where the documents are selected because they contain or do not contain the specified words.
  - **Wildcard** search, using front, middle or end masking for words and characters.
- Search for documents written in various languages. If a linguistic index was defined, a different type of linguistic processing will be used, depending on the document′s language. Some of these types are:
  - Word and sentence separation
  - De-hypenation
  - Sentence-begin processing
  - Normalization of terms to a standard form, with no capital letters and changing accented letters to a form without accents

- Reducing terms to their base form. For example, ″mice″ is reduced to ″mouse″
- Decomposition of compound words
- Stop-word filtering eliminating insignificant words from the index
- Part-of-speech filtering where only nouns, verbs and adjectives are indexed

6.2.2 Supported Environments

The DB2 Text Extender is client/server software; so part of the product is installed on the client, and another part is installed on the server.

- The main part of the DB2 Text Extender, which is called the Text Extender Server, is installed in the same machine as the DB2 Server product.
- The client part of the DB2 Text Extender, that is the Text Extender Utilities, must be installed on a machine which has the DB2 Client product on it. The DB2 Text Extender Client communicates with the DB2 Text Extender Server via the DB2 client connection.

The currently supported server platform for the DB2 Text Extender is AIX, while the client components can be installed either on AIX or OS/2 clients.

Figure 73 shows a simple configuration of the DB2 Text Extender environment.

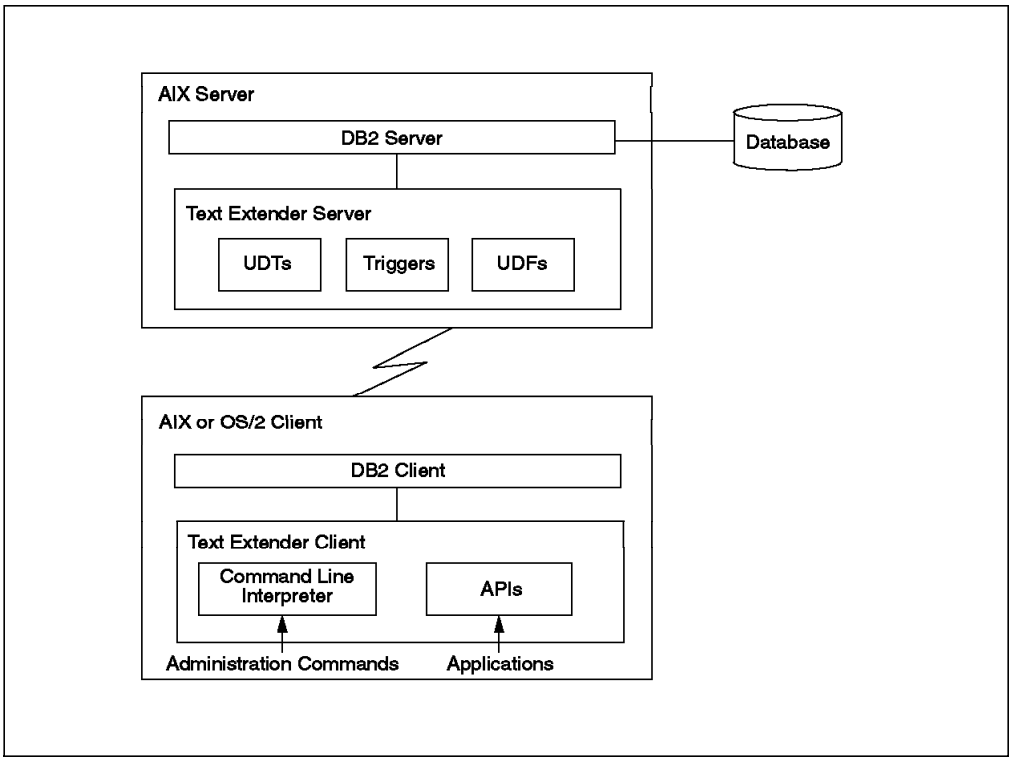


Figure 73. A Simple Configuration of the DB2 Text Extender

6.3 The DB2 Text Extender Installation

The DB2 Text Extender is included with the DB2 Cross-Industry Extender products; however, you need to use the DB2 Cross-Industry Extenders package to install the DB2 Text Extender.

To install the DB2 Text Extender in AIX, you can use the System Management Interface Tool (SMIT):

1. If you are going to install the Text Extender Server, login as *root* at the server where you want to install the product.
2. Enter `smit` or `smitty` at the command prompt.
3. Select the **Software Installation and Maintenance** option.
4. Select **Install and Update Software** from the Software Installation and Maintenance screen.
5. Select **Install/Update Selectable Software (Custom Install)** from the Install and Update Software screen.
6. Select **Install Software Products at Latest Level** from the Install/Update Selectable Software (Custom Install) screen.
7. Select **Install New Software Product at Latest Level** from the Install Software Products at Latest Level screen.
8. Type your input device name on the entry fields, and press **Enter**. You can press **F4** to see a list of devices available on your machine.
9. From the list of products available in db2tx_01_01_0000, select:
   - Browser, if you are planning to use the DB2 Text Extender browser
   - Common Modules for Client and Server
   - English (US) Documentation
   - Language dictionary and stop words, in the language you prefer
   - Search Service for DB2/6000 Text Extender
10. After you make your selections, press **Enter** to install the product. When you are prompted for confirmation of the installation, press **Enter** to confirm. The installation will take a few seconds.
11. If you are installing the client components, you can install more than one dictionary. These dictionaries are installed automatically when you install the server components. To install a dictionary at the client, repeat steps 2 through 8, and select the language dictionary you need from the dictionaries available on Table 21. Installing a dictionary will automatically install the stop words for this language.

Table 21 (Page 1 of 2). DB2 Text Extender Dictionaries

Dictionary	Language
db2tx_01_01_0000.dic.Da_DK	Danish
db2tx_01_01_0000.dic.NI_NL	Dutch
db2tx_01_01_0000.dic.En_GB	English (U.K.)
db2tx_01_01_0000.dic.En_US	English (U.S.)
db2tx_01_01_0000.dic.Fi_FI	Finnish

Table 21 (Page 2 of 2). DB2 Text Extender Dictionaries	
Dictionary	**Language**
db2tx_01_01_0000.dic.Fr_FR	French
db2tx_01_01_0000.dic.Fr_CA	French (Canada)
db2tx_01_01_0000.dic.De_DE	German
db2tx_01_01_0000.dic.De_CH	German (Swiss)
db2tx_01_01_0000.dic.Is_IS	Icelandic
db2tx_01_01_0000.dic.It_IT	Italian
db2tx_01_01_0000.dic.No_NO	Norwegian
db2tx_01_01_0000.dic.Pt_PT	Portuguese
db2tx_01_01_0000.dic.Es_ES	Spanish and Catalan
db2tx_01_01_0000.dic.Sv_SE	Swedish
db2tx_01_01_0000.dic	All supported language

12. If you want to install the online information for the DB2 Text Extender, repeat the steps 2 through 8, and select **db2tx_01_01_0000.doc.En_US**.

6.3.1 Setting Up the DB2 Text Extender

After the server product is installed on your machine, do the following:

1. **Establish the DB2 Cross-Industry Extenders instance**.

   To do this, you need to perform the following:

   a. Login as *root*.

   b. Go to the instance directory of the DB2 Cross-Industry Extenders product.

      ```
      cd /usr/lpp/dmb_10/instance
      ```

   c. Run `dmbinstance`.

      ```
      ./dmbinstance instance ID
      ```

      where `instance ID` is an existing DB2 instance user ID or a user ID that will be associated with a DB2 instance.

      The DB2 Cross-Industry Extenders will create the dmb directory under /u/instanceid directory. You should not create any files under the /dmb directory since this directory will be deleted whenever you remove the instance.

      While running this command, you will be prompted for the following:

      - If you want to make the instance ID a DB2 instance owner. This question will be prompted only if the instance ID supplied is not a DB2 instance owner. If you do not want instance ID to be a DB2 instance owner, it cannot be a DB2 Text Extender instance owner.
      - If you want to create an instance for the DB2 Cross-Industry Extenders for the specified instance ID. Answer yes if you are installing or planning to install later, all the products in the DB2 Cross-Industry Extenders.
      - If you want to create an instance for the DB2 Text Extender for the specified instance ID. It is the same as using the `db2txinstance` command to create an instance only for the DB2 Text Extender.

- If you want to create an instance for a previous DB2 Fingerprint Extender installation. Your answer will not affect the installation of the DB2 Cross-Industry Extenders.

d. Logout.

2. **Establish the instance environment for DB2 Cross-Industry Extenders**

   To set up the instance environment, perform the following tasks:

   a. Log in as instance ID

   b. Edit the login profile of the instance ID:

      ```
      vi .profile
      ```

      Add the following lines to the login profile:

      - If you are using the C shell:

        ```
        setenv LANG En_US
        source dmb/dmbcshrc
        ```

      - If you are using the Korn or Bourne shell:

        ```
        export LANG=En_US
        . dmb/dmbprofile
        ```

      The dmbcshrc and dmbprofile are shell scripts that contain statements used to set the path in your operating system and to set the environment variables for the DB2 Cross-Industry Extenders use.

      There is also a db2txprofile file where you can set up the DB2 Text Extender environment variables. If you are using the DB2 Text Extender, you can choose to run the db2txprofile; if not, the dmbprofile has a line pointing to run db2txprofile. Refer to 6.3.2, “Environment Variables” on page 117 for detailed information about the DB2 Text Extender environment variables.

   c. Logout

3. **Create a local search service**

   You need to perform this step only when installing the DB2 Text Extender Server. To create the local search service, do the following:

   a. Login as *instanceid*

   b. Enter this command:

      ```
      DB2TXCFG ssrv-name port max-tasks avail-tasks time
      ```

      where

      ssrv-name The search service name, which is the *instanceid* in uppercase letters.

      port The TCP/IP port used by the clients to make a connection to this search service. It must be the same port used by DB2 clients to connect to the database.

      - If you were creating a Text Extender instance for a user who is not a DB2 instance owner, you must have been asked to create a DB2 instance for this user. At this time, the definition of a TCP/IP service is added at the end of the etc/services file. Use the port number corresponding to this service.

- If the user was a DB2 instance owner, use the TCP/IP service port number defined for this instance in the etc/services file.

max-tasks The maximum number of tasks that can be handled by the service at the same time. The valid value is a number between 1-100. The higher you set this value, the slower the system runs. But if you set this value too low, not all the clients can make a search at the busy time. So you have to consider the number of clients you have and the percentage of that number which may do a search at the same time.

avail-tasks The number of tasks to be started in advance to handle future requests. The valid value is a number between 1-10. The value for this parameter must be lower or equal to the number set for *max-tasks*.

time The time that a search service task remains occupied without receiving any request from a client. The valid value is a number between 100-99900. After this time, the connection to the client is terminated so other clients can use the service. If you consider the queries are often complex, set the time to 600 or more. Otherwise, you can set it to 100.

To get help information about this command, enter the command:

```
db2txcfg
```

To display the current configuration, enter the command:

```
db2txcfg -d
```

After installing the client product in your machine, you can connect to the server using the DB2 Client product and run your applications using the DB2 Text Extender Utilities.

6.3.2 Environment Variables

The environment variables for DB2 Text Extender are defined in the following files:

- db2txcshrc for the C shell
- db2txprofile for the Korn shell and the Bourne shell

These files are located in the /home/$DB2TX_INSTOWNER/db2tx directory. Some administration commands require parameters that are mentioned in these files. These files set the default values of some parameters to be used by the DB2 Text Extender. If you are submitting the commands without specifying the value of the parameters, DB2 Text Extender will use the default value from these files.

To use these environment variables files, you can run them at the command prompt. But it is recommended for your convenience to run the files automatically during login by adding the shell script on your login profile (refer to 6.3.1, “Setting Up the DB2 Text Extender” on page 115). .

The environment variables are the following:

1. **DB2TX_INSTOWNER**

   This is the login name of the AIX user that owns the instance.

2. **DB2TX_INSTOWNERHOMEDIR**

   This is the home directory of the instance owner.

3. **DB2TX_CCSID**

   This is the default Coded Character Set Identifier that is used to determine the codepage of the text document. It must be the same used by the database.

   To determine the codepage being used by a DB2 database, you should look at the database configuration. This can be displayed using the command:

   ```
   db2 get database configuration for <database-name>
   ```

   The currently supported codepages are:

   819 Latin

   850 Latin

   813 Greece

   874 Thailand

   912 Czech, Croatia, Hungary, Poland, Servia (Latin), Slovenia, Slovakia

   915 Bulgaria, FYR Macedonia, Servia (Cyrillic), Russia

   920 Turkey

   1046 Arabia

   1089 Arabia

4. **DB2TX_FORMAT**

   This is the default format of the text documents that is needed for the index. The initial setting is TDS; that is flat ASCII. The current supported formats are:

   TDS Flat ASCII

   AMI Ami Pro Architecture Version 4

   FFT IBM Final Form Text: Document Content Architecture

   RFT IBM Revisable Form Text: Document Content Architecture

   RTF Microsoft Rich Text Format Version 1

   WP5 WordPerfect (OS/2 and Windows) Versions 5.0, 5.1 and 5.2

   MSWORD Microsoft Word Versions 5.0 and 5.5

5. **DB2TX_LANGUAGE**

   This is the default dictionary name. The initial setting is US_ENGLISH. Other dictionaries provided by the DB2 Text Extender are:

BRAZILIAN	CAN_FRENCH	CATALAN
DANISH	DUTCH	GERMAN
FINNISH	FRENCH	ICELANDIC
ITALIAN	BM_NORWEGIAN	NN_NORWEGIAN
BMNN_NORWEGIAN	SWEDISH	PORTUGUESE
SWISS_GERMAN	SPANISH	UK_ENGLISH

6. **DB2TX_INDEXDIR**

   This is the directory that will be used to store the index. The initial setting is /home/$DB2TX_INSTOWNER/db2tx/indices.

7. **DB2TX_INDEXTYPE**

   This is the default index type to be used if you do not specify it when creating the index. The initial setting is LINGUISTIC.

8. **DB2TX_UPDATEFREQ**

   This is the setting for the frequency of the index update. The initial setting is NONE, and you must execute the UPDATE INDEX command each time you want the index to be updated.

   Below is an example of using the frequency setting for the index update:

   ```
   min(50) d(1,3,5) h(12,17) m(0)
   ```

   The index will be updated every Monday, Wednesday, and Friday at 12:00 (noon) and 17:00 (5 p.m.) only when there would is minimum of 50 text documents to be indexed recorded in the log table (refer to 6.4.1, “Maintaining the Text Index” on page 120).

9. **DB2TX_UPDATEINDEX**

   This is the variable to determine whether the first index is done immediately after the index is created (UPDATE) or later, according to the frequency setting of the index update (NONE). The initial setting is UPDATE.

6.3.3 The SAMPLE Table

The DB2 Text Extender provides a script to create a SAMPLE table to test the product. This script is create_sample, and it is located in the /home/$DB2TX_INSTOWNER/db2tx/samples directory. To run this script, enter the following:

```
create_sample database
```

where database is the name of a DB2 database existing in the DB2 Server. If you do not specify the database name, the command will use the default database defined by the DB2DBDFT parameter in the db2profile file.

The create_sample script will do the following:

1. Connect to the database
2. Create the DB2TX.SAMPLE table
3. Enable a text column named COMMENT on DB2TX.SAMPLE
4. Create the index

This table is used in the example programs presented in 6.8, “Examples” on page 139.

6.4 How the DB2 Text Extender Works

In order to perform quick and efficient searching, there are several things that DB2 Text Extender creates and maintains. This section describes what they are and what are they used for.

6.4.1 Maintaining the Text Index

Before you can use the DB2 Text Extender with your databases, you must enable your database, your table, and your text column to be used by the DB2 Text Extender. These are some of the administration tasks that you must perform as explained in 6.5, “Administration Tasks” on page 123.

At the time you enable the database, table, and column, the DB2 Text Extender creates the following tables, views and columns, which are used to maintain the text index.

- **Catalog View**

  The Catalog view created by the DB2 Text Extender contains the information about the tables and columns that are enabled for the Text Extender. The catalog view is called TEXTCOLUMNS and is needed to process the user requests.

  You can use SQL statements to view data in the Catalog view, but you cannot modify it. Data in the Catalog view is updated as follows:

  – New entries will be added when you enable new tables or columns.

  – Existing entries are updated when you change the index setting using the `CHANGE INDEX SETTING` command.

  – Existing entries will be removed whenever you disable tables or columns to be used by the DB2 Text Extender.

- **Log Table**

  The Log table is required to keep the information about the changes of text documents in an enabled table. The extender uses triggers to update log tables whenever text documents are added, updated or deleted from the table. This ensures that these documents will be indexed at the next indexing time.

- **Handle Column**

  A handle column is a column created by the DB2 Text Extender. It is added to the table that has the enabled column. Each enabled column has one handle column which contains the text handles associated to the enabled column. The information stored in a text handle is:

  – The document ID

  – The name and location of the associated index

  – The document information: the format, the CCSID and the language

6.4.2 Indexing

Indexing is something important to be considered in an information retrieval system since it allows the system to perform queries more efficiently. The system will not search through the whole text of documents, just the index.

A text index is a list that consists of significant words extracted from the text documents. Each word is stored together with the information about the document that contains it. Insignificant words, such as ″and″, ″the″ and ″of″, are not indexed. To avoid these words from being indexed, DB2 Text Extender has a list called *stop words* which consists of all the insignificant words.

There are two steps involved in the indexing process:

1. Recording text documents that need to be indexed in a log table

   This is an automatic process done by DB2 triggers whenever you insert, update or delete a text document in a column.

2. Indexing text documents listed in the log table

   This step is performed periodically. In this step, the significant words of the documents inserted or changed in the column are added to the index, and the words of those documents that were deleted from the column are removed from the index.

6.4.2.1 Multi-index and Common-Index Tables

Something that needs to be considered before enabling the text table to be used by the DB2 Text Extender is whether to use a *multi-index table* or a *common-index table*.

Multi-index Table

A multi-index table is a table which has a separate text index for each text column. Using multi-index table gives you flexibility because you can create different types of indexes for each text column, set a different periodic time for updating the index, and create different directories for storing the index. Since index processing may consume a large amount of time and resource, having a multi-index table allows you to spread this activity over a period of time by setting different times for updating each index.

Figure 74 shows the indexes and log tables for a multi-index table.

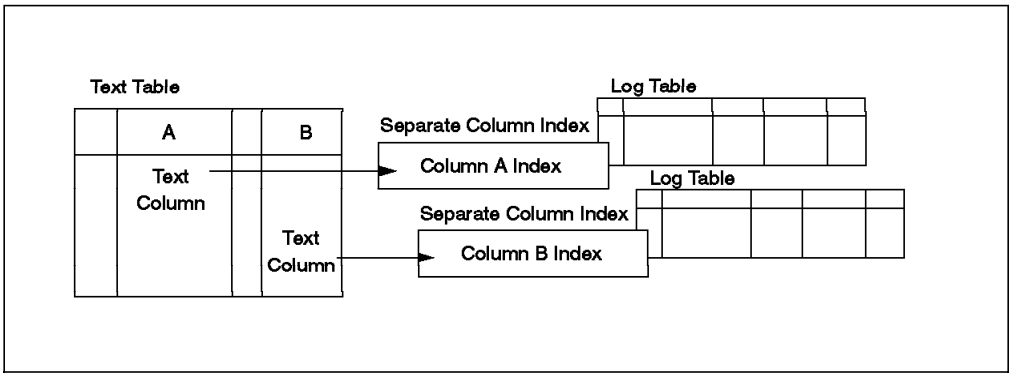


Figure 74. Multi-Index Table: A Separate Text Index for Each Text Column

Common-index Table

A common-index table is a table which has one text index common to all the text columns in that table. The advantage of a common-index table is that it is easier to maintain. If you have to change the index characteristic, you only need to make the changes once.

Figure 75 on page 122 shows the index and log table for a common-index table.

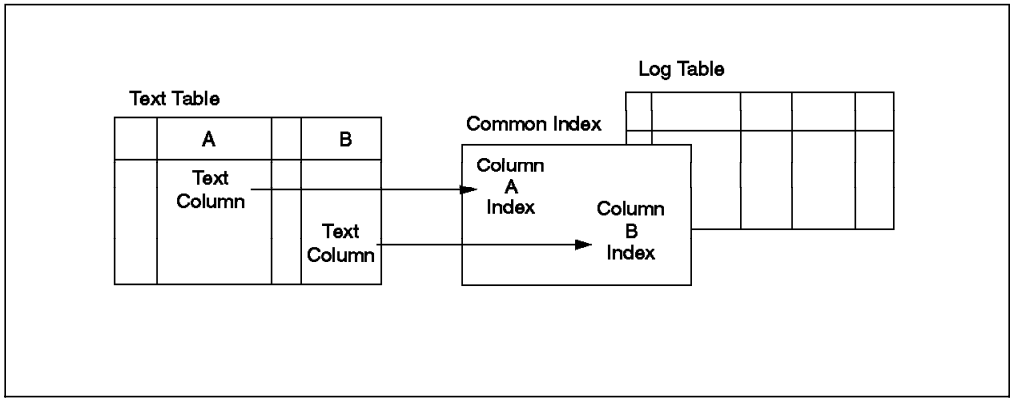


Figure 75. Common Index Table: A Common Index for All the Text Columns

6.4.2.2 Types of Indexes

The DB2 Text Extender provides three types of indexes. You need to know these types of indexes before you perform the ENABLE TEXT COLUMN command discussed in 6.5, “Administration Tasks” on page 123. You can change the type of index currently used at any time if you think that it is not suitable anymore. When you change the type of an index on a column, the DB2 Text Extender performs the following tasks:

1. Deletes the existing index
2. Creates a new empty index
3. Adds the entries for all the text documents in the column to the log table for re-indexing

The index update will be done accordingly with the DB2TX_UPDATEFREQ and DB2TX_UPDATEINDEX variables.

The three types of indexes provided by the DB2 Text Extender are:

1. **Linguistic Index**

   In a linguistic index, words are reduced to their base form before being stored in an index. This linguistic processing is also applied when a user queries against a linguistic index. It means that the search word is also reduced to its base form.

   For example, to store in the index the word ″mice″, it is reduced to ″mouse″. And when you submit a query using ″mice″ as the search word, it is also reduced to ″mouse″ before the search begins.

   Some advantages of using the linguistic index are:

   - It requires the least amount of disk space.
   - You can search using any variations of a word, which will give you the same results.

   The linguistic index needs more time to do indexing and searching as compared with a precise index because of the previous process of reducing the word to its base form.

2. **Precise Index**

   In a precise index, words are indexed exactly as they appear in the text documents. It means that they are case-sensitive. The same linguistic

process is applied to the search words in queries, which will give results only for the exact same words.

The advantages of using a precise index are:

- You can use masking characters in the queries which will broaden the search. For example, the search word ″comp*″ will find ″compress″, ″compiler″ and so on.
- You can be more precise in your search.
- The indexing and retrieval is faster because there is no reduction process.

This type of index requires more disk space for the index because every word that has a different form is indexed separately.

3. **Dual Index**

   A dual index is a combination of linguistic index and precise index. Each word is indexed in three forms:

   - **Normalized form** — a standard form of words in all lower-case letters and without accents.
   - **Base form** — the infinitive form of words.
   - **Precise form** — the exact form of words as they are in the text documents.

   The main advantage of using a dual index is its flexibility. You can choose to use either a linguistic or precise search in the query statements.

   Because each word is indexed in three forms, this type of index requires the most amount of disk space and is the slowest for indexing and searching.

6.5 Administration Tasks

We can divide the administration tasks into those that are used for the administration of the Text Extender Server and those used for the Text Extender Client.

6.5.1 Administration of the DB2 Text Extender Server

You can issue the administration commands from the operating system prompt.

Basically, the administration of the server consists in performing the following tasks:

1. Start the Text Extender server:

   a. Log in using the user ID of the DB2 Text Extender instance owner (refer to the DB2TX_INSTOWNER variable in 6.3.2, “Environment Variables” on page 117).

   b. Enter the following at the command prompt:

      ```
      db2txstart password
      ```

      where *password* is the password of the DB2 Text Extender instance owner user ID.

      When the DB2 Text Extender Server start is completed, you will receive the message:

      ```
      The requested operation completed successfully.
      ```

2. Display the status of the DB2 Text Extender Server:

```
db2txstatus
```

If the DB2 Text Extender Server is already started, you will get a screen similar to Figure 76.

```
$ db2txstatus

        SearchManager status for service: DB2

        SearchManager controller is running

        SearchManager administration task running:  no

        SearchManager communication service is running

        SearchManager communication details
                0 client(s) busy.
                10 client(s) ready for connection.
                0 client(s) not yet created.
The requested operation completed successfully.
```

Figure 76. Status Screen When the DB2 Text Extender Server is Up

But if the DB2 Text Extender has not been started yet, you will get the messages shown in Figure 77.

```
$ db2txstatus

        SearchManager status for service: DB2

        SearchManager controller is not running

        SearchManager communication service is not running
The requested operation completed successfully.
```

Figure 77. Status Screen When the DB2 Text Extender Server is Not Running

3. Stop the DB2 Text Extender Server:

```
db2txstop
```

When the DB2 Text Extender Server stop is completed, you will receive the message:

```
The requested operation completed successfully.
```

6.5.2 Administration of the DB2 Text Extender Client

The tasks involved in the administration of the DB2 Text Extender Vlient are:

- Preparing the text documents for searching, such as enable a database, a table and a column to be used by the DB2 Text Extender.
- Maintaining the index, such as update, recreate or change the settings of an index.
- Getting information, such as the information about index settings, text settings for a column, environment variables settings, and the enabled status of databases, tables and columns.
- Working with the DB2 Text Extender Catalog views.

- Releasing text that has been prepared to be used by the DB2 Text Extender, such as disable a column, a table or a database.

Before you enter any administration command, be sure that the DB2 Text Extender Server is already running. The administrator can check it by entering db2txstatus as mentioned in 6.5.1, “Administration of the DB2 Text Extender Server” on page 123.

As in DB2, you can submit the administration commands either from the operating system command prompt by prefixing the command with db2tx or from the Command Line Processor provided by the DB2 Text Extender. To start the DB2 Text Extender Command Line Processor, you enter the following at the operating system command prompt:

```
db2tx
```

The DB2TX prompt will be displayed, and all the commands entered from this prompt will be interpreted as DB2 Text Extender commands.

```
db2tx⇒
```

Enter quit to exit the DB2 Text Extender Command Line Processor.

Because the administration tasks involve a lot of indexing process, you need to make your decisions about:

- The type of index you are going to use
- Whether to use a multi-index table or a common-index table
- The directory where you are going to store the index
- The Coded Character Set Identifiers (CCSID), the languages, and the formats of the text in which you intend to search

You can check the default values of type of index, index directory, CCSID, language, and format of text on the db2txprofile file (refer to 6.3.2, “Environment Variables” on page 117). You do not need to specify these values in your commands if they are already defined as the default values.

6.5.3 Administration Commands Summary

Table 22 lists the available commands for the DB2 Text Extender administration.

Table 22 (Page 1 of 3). The Administration Commands Summary

Command	Purpose	Note
?	Command Line Processor help.	
CONNECT TO	Connects to a database.	Without submitting this command, the DB2 Text Extender will try to connect to the default database specified by the DB2DBDFT environment variable.
ENABLE DATABASE	Prepares a database to be used by the DB2 Text Extender.	The DB2 Text Extender will: • Declare UDFs and UDTs to DB2 • Create a catalog view called TEXTCOLUMNS

Table 22 (Page 2 of 3). The Administration Commands Summary

Command	Purpose	Note
ENABLE TEXT TABLE	Prepares a text column for use by Text Extender. Run this command if you want to use common index. Skip this command if you want to use multi-index.	The DB2 Text Extender will: • Create an empty text index • Create an empty log table • Create triggers to update the log table Specify the type of index, the frequency of updates and the directory to store the index; otherwise the DB2 Text Extender will use the default values on db2txprofile.
ENABLE TEXT COLUMN	Prepares a column to be used by the DB2 Text Extender.	The DB2 Text Extender will: • Create an index • Create a log table • Set the document information, such as the format, the CCSID and the language • Add a handle column to the table Specify the type of index, the frequency of updates and the directory to store the index (if you are using multi-index); otherwise the DB2 Text Extender will use the default values on db2txprofile.
UPDATE INDEX	Updates an index immediately.	Use this command to update an index without waiting for the next periodic indexing time. Specify the handle-column name for updating column in a multi-index table.
RECREATE INDEX	Recreates a damaged text index.	The DB2 Text Extender will: • Delete an existing index • Create a new index The indexing can begin immediately or when the next schedule for periodic indexing time was specified, according to the UPDATEINDEX keyword.
CHANGE INDEX SETTING	Changes the characteristics of an index.	You can change either the index type, the update frequency or the directory name where the index is stored. If you change the index type or the directory name, the DB2 Text Extender will: • Delete an existing index • Create a new index
GET STATUS	Displays the enabled status of databases, tables and columns.	
GET ENVIRONMENT	Displays the current settings of the DB2 Text Extender environment variables.	
GET INDEX SETTINGS	Displays the characteristics of an index.	
GET TEXT INFO	Displays the text settings of a text column.	
DISABLE TEXT COLUMN	Disables a text column to be used by the DB2 Text Extender.	The DB2 Text Extender will: • Delete the index for this column (if it uses a multi-index) • Delete the log table (if it uses a multi-index) • Delete triggers used to maintain the log table • Set the contents of the handle column to null

Table 22 (Page 3 of 3). The Administration Commands Summary

Command	Purpose	Note
DISABLE TEXT TABLE	Disables a table to be used by the DB2 Text Extender.	The DB2 Text Extender will: • Delete the index (if it uses a common-index) or delete all the indexes for the text columns in this table (if it uses multi-index) • Delete the common log table (if it uses a common-index) or delete all the log tables (if it uses a multi-index) • Delete the triggers used to maintain the deleted log tables • Set the contents of the handle columns to null
DISABLE DATABASE	Disables a database to be used by the DB2 Text Extender.	The DB2 Text Extender will: • Delete the catalog view created for this database • Delete the declaration of the DB2 Text Extender's UDTs and UDFs from this database • Delete all the indexes for the text tables or text columns in this database • Delete the log tables and triggers • Delete the contents of all the column handles

The ENABLE TEXT COLUMNS supports only VARCHAR, LONG VARCHAR or CLOB text columns to be used by the Text Extender. If your text column is in a different data type, such as a User-Defined Type, you need to create a User-Defined Function to convert the User-Defined Type used in your text column.

The User-Defined Function should have your User-Defined Type as its input and VARCHAR, LONG VARCHAR or CLOB type as its output.

To enable the column, you need to put the FUNCTION parameter in the ENABLE TEXT COLUMNS statement. For example:

```
db2tx "ENABLE TEXT COLUMN table-name text-column-name
      FUNCTION function-name
      HANDLE handle-column name
      ...
```

where:

table-name The name of your table

text-column-name The text column to be enabled

function-name The User-Defined Function used to convert the text column from the unsupported data type to the supported data type

handle-column-name The handle column name

6.6 UDTs and UDFs

In this section, we discuss the UDTs and UDFs that are provided by the DB2 Text Extender and show some examples of how to use these functions. To clarify the examples in this section, the structure of the table we use in the examples is shown below.

```
CREATE TABLE db2tx.sample (
       forum       VARCHAR(30),
       date        TIMESTAMP,
       author      VARCHAR(50),
       subject     VARCHAR(100),
       ref_date    TIMESTAMP,
       ref_author  VARCHAR(100),
       comment     LONG VARCHAR)
```

This is the table sample that you might have already created after installing the DB2 Text Extender, as explained in 6.3.3, “The SAMPLE Table” on page 119.

6.6.1 User Defined Types

The DB2 Text Extender provides specific UDTs used in some of the DB2 Text Extender UDFs:

1. **DB2TEXTH**

   The source data type of DB2TEXTH is VARCHAR(60) FOR BIT DATA.

   This UDT is used as the data type for the handle columns. It is a variable-length string that contains the information needed for indexing the text document. The information stored in it is:

   - A document ID
   - The name of the server where the text is to be indexed
   - The name of the index
   - Information about the text document

   The INIT_TEXT_HANDLE and HANDLE functions return this data type.

2. **DB2TEXTHLISTP**

   The source data type of DB2TEXTHLISTP is VARCHAR(16) FOR BIT DATA.

   DB2TEXTHLISTP is a text handle list pointer. That is a pointer to a list of text handles associated with text documents found by a search.

   The HANDLE_LIST function returns this data type.

6.6.2 User-Defined Functions

The DB2 Text Extender also provides UDFs. To use these UDFs in SQL statements, you must add “DB2TX” to the function path. Issue the following command from the DB2 Command Line Processor:

```
SET CURRENT FUNCTION PATH = "DB2TX", current function path
```

where current function path is the function path already set up before you submit this command. You can write “current function path” for this parameter.

To see the current function path setup, enter the following using the DB2 Command Line Processor:

```
VALUES CURRENT FUNCTION PATH
```

You need to set the current function path each time you connect to a database. An alternative way to use UDFs without having to set the current function path is by explicitly specifying the qualifier name of the function. For example, using the DB2TX.CONTAINS function instead of CONTAINS.

All the examples of SQL statements in this section are invoked from the interactive mode of CLP. Otherwise, the parameters written between quotation marks must be written as follows:

```
For (..., '"compress"') write (..., '\"compress\"')
```

The following are the UDFs that are provided by the DB2 Text Extender and some examples of how to use these functions within SQL statements.

1. **CCSID**

   This function is used to get the CCSID from a text handle which is set for each column when you enable a text column.

   For example:

   ```
   SELECT distinct CCSID(commenthandle) FROM db2tx.sample
   ```

   The result is:

   ```
   1
   -----------
           850

     1 record(s) selected.
   ```

   It means that the CCSID used for the text handle “commenthandle” is 850.

2. **CONTAINS**

   This function is used to search for text within the documents. It will return an integer value of 1 if the document contains the text and 0 if the document does not contain the text.

   The following SQL statement shows you how to use the CONTAINS function:

   ```
   SELECT date, subject \
   FROM db2tx.sample \
   WHERE contains (commenthandle, '"compress"')=1
   ```

   This SQL statement will give you a list of the dates and subjects of all the documents that have the term “compress” in the text referred by the handles in the column COMMENTHANDLE, as follows:

   ```
   DATE                       SUBJECT
   -------------------- ----------------------------------------------------
   07-28-16.58.53.000000 numerous questions ...
   07-28-19.20.48.000000 numerous questions ...

    2 record(s) selected.
   ```

3. **FORMAT**

   Use this command to do one of the following:

   a. To get the format of the document specified in a text handle.

      ```
      SELECT distinct format(commenthandle) \
      FROM db2tx.sample
      ```

   b. To change the format specification in a document’s text handle.

      For example:

      ```
      UPDATE db2tx.sample \
      SET commenthandle=format(commenthandle,'AMI')
      ```

4. **HANDLE**

Returns a text handle selected by sequence number from a list of text handles. Its data type is DB2TEXTH.

For example:

```
SELECT HANDLE(HANDLE_LIST(commenthandle,'"compress"'),1) \
FROM db2tx.sample \
WHERE CONTAINS (commenthandle,'"compress"')=1
```

This function is used mostly with HANDLE_LIST, NO_OF_MATCHES and RANK to perform complex SQL statements which can improve the performance of the text searches. Refer to *Improving search performance* in *Database 2 Text Extender: Administration and Programming*.

5. **HANDLE_LIST**

   This function is used to search for text documents using a search term. The returned value is a DB2TEXTHLISTP data type that points to a list of text handles for the found documents. The list will be empty if there is no text document found. You can use the NO_OF_DOCUMENTS function to find out whether the list is empty or not.

   For example, to get the pointer to the list of the text handles for the documents containing the word "compress":

   ```
   SELECT HANDLE_LIST(commenthandle,'"compress"') \
   FROM db2tx.sample \
   WHERE CONTAINS (commenthandle,'"compress"')=1
   ```

   This pointer is accessible only within the scope of the SQL statement using this function.

   If you run this statement from the CLP, you will get the following:

   ```
   1
   ------------------------------
   x'312018C0D8000000022018C378047900'
   x'312018C0D8000000022018C378047900'

     2 record(s) selected.
   ```

6. **INIT_TEXT_HANDLE**

   This function is used to set the format and language to values that are different from the previous format and language set by the default environment variables or set when you enable a text column.

   For example, you enable a column with the following settings:

   ```
   ENABLE TEXT COLUMN db2tx.sample \
   COMMENT HANDLE commenthandle \
   INDEXTYPE dual \
   LANGUAGE us_english \
   FORMAT tds
   ```

   But then, you want to insert a new data which is not using US_ENGLISH as its language nor TDS as its format. So, you insert the data using the following command:

   ```
   INSERT INTO db2tx.sample \
   (forum, comment, commenthandle) \
   VALUES ('db2tx.cforum', 'If you have a problem ...', \
   init_text_handle('AMI', 'FRENCH')
   ```

7. **LANGUAGE**

The LANGUAGE function is used to perform the following tasks:

- To get the language of the document specified in a text handle.

  For example:

  ```
  SELECT distinct language(commenthandle) FROM db2tx.sample
  ```

- To change the language specification in a document's text handle and to get the changed text handle.

  For example:

  ```
  UPDATE db2tx.sample \
  SET commenthandle=language(commenthandle,'FRENCH')
  ```

8. **NO_OF_DOCUMENTS**

   This function is used to get the number of items in a list of text documents found by a search. It returns an integer value which is the number of entries found in a list of text handles.

   The parameter of this function is an expression with a DB2TEXTHLISTP data type, which is returned by calling the function HANDLE_LIST. The list result from the HANDLE_LIST function exists only within the scope of an SQL statement; so you have to use this function in the same SQL statement as the HANDLE_LIST function.

   For example, to know how many documents contain the word "compress",

   ```
   SELECT NO_OF_DOCUMENTS(HANDLE_LIST(commenthandle,'"compress"')) \
   FROM db2tx.sample \
   WHERE CONTAINS(commenthandle,'"compress"')=1
   ```

9. **NO_OF_MATCHES**

   Depending on the input parameters, NO_OF_MATCHES has two different functions:

   - To determine how many matches are found in a text document. The function will return an integer value. For example:

     For example, to select the documents which contain the word "compress" one or more times:

     ```
     SELECT date, subject \
     FROM db2tx.sample \
     WHERE no_of_matches(commenthandle, '"compress"') > 0
     ```

   - To return the number of matches from a handle in a list of text. This function is not yet supported.

10. **RANK**

    Depending on the input parameters, RANK has two different functions:

    - To determine the rank value of a text document that has been returned by a search. The function will return a DOUBLE value between 0 and 1, indicating the number of matches found in the document in relation to the document's size.

      For example:

      ```
      SELECT date DATE, rank(commenthandle,'"compress"') RANK \
      FROM db2tx.sample ORDER BY RANK
      ```

    - To get the rank value from a handle in a list of text handles. This function is not yet supported.

11. **REFINE**

    When a search finds too many occurrences, it can be useful to narrow or *refine* the search by combining the initial search argument with a second one in a Boolean AND relationship. With this function, you can take the two search arguments and get a LONG VARCHAR data type value consisting of the two input parameters connected by the AND operator.

    Before using the REFINE function, you need to perform the following two steps:

    a. Create a table for the old search arguments:

    ```
    CREATE TABLE previous_searches \
     (step INT, \
      searchargument LONG VARCHAR)
    ```

    b. Search with the first search argument, "compress" for example:

    ```
    SELECT comment \
     FROM db2tx.sample \
     WHERE CONTAINS (commenthandle,'"compress"')=1
    ```

    And insert this search argument into the table PREVIOUS_SEARCHES:

    ```
    INSERT INTO previous_searches \
     VALUES (1,'"compress"')
    ```

    Now, you are ready to refine the search. Assume that you want to refine the search by combining the previous search term with the word "compiler":

    ```
    WITH LAST_STEP(step_max) \
     AS (SELECT MAX(step) \
        FROM previous_searches),
     LAST_SEARCH(last_search)
     AS (SELECT searchargument \
        FROM previous_searches, last_step \
        WHERE step = step_max)
     SELECT comment \
     FROM db2tx.sample, last_search \
     WHERE CONTAINS (commenthandle, \
           REFINE (last_search,'"compiler"'))=1
    ```

    You can insert the refined search argument into the PREVIOUS_SEARCHES table for use by further searches.

    Another way to refine search results without using the REFINE function is by storing the result in a temporary table and making the new search against this table. However, depending on the number of qualifying terms, this method may be less efficient.

6.6.3 Search Arguments

Some functions, such as CONTAINS, NO_OF_MATCHES, RANK, and HANDLE_LIST, require search arguments. There are several ways to specify a search argument in a function, they are:

1. Searching for several terms.

   You need to combine the terms you are searching for, using commas.

   For example:

   ```
   CONTAINS(commenthandle, '("compiler", "DB2", "compress")')=1
   ```

2. Searching using Boolean operators AND, OR and NOT.

You use Boolean operators in your search arguments. For example:

```
CONTAINS(commenthandle, '"compress" | "compiler"')=1
```

There are some considerations when using more than one Boolean operator in a search argument. The DB2 Text Extender evaluates Boolean operators from left to right, but the logical AND (&) operator binds stronger than the logical OR (|) operator.

For example:

```
"DB2" & "compiler" | "compress" & "zip"
```

will be evaluated as:

```
("DB2" & "compiler") | ("compress" & "zip")
```

You must use parentheses to change the order, for example:

```
"DB2" & ("compiler" | "compress") & "zip"
```

The NOT operator is used to exclude particular terms from the search.

For example:

```
("compress", "compiler") & NOT "DB2"
```

The NOT operator, cannot be used with IN SAME SENTENCE AS, IN SAME PARAGRAPH AS or SYNONYM FORM OF clauses.

3. Searching for a term in a dual index.

   In a dual index, you can search for the exact form of a word or its variations. Variations of one term can be its plural form, its past tense form, and so on. To determine whether you want to search using the exact form or the variations, you need to specify the following:

   To search for the occurrences of "utility":

   ```
   CONTAINS(commenthandle, 'PRECISE FORM OF "utility"')=1
   ```

   To search for variations of "utility":

   ```
   CONTAINS(commenthandle, 'STEMMED FORM OF "utility"')=1
   ```

4. Searching using character-masking.

   The DB2 Text Extender uses two-character masking:

   Underscore (_) Which represents one character in a search term

   Percent (%) Which represents any number of arbitrary characters

   When you want to search for a term that includes one of the masking characters, you need to use an *escape* character. Specify it using the ESCAPE keyword.

   For example, to find the term "10% interest" using "!" as the escape character:

   ```
   CONTAINS(commenthandle, '"10!% interest" ESCAPE "!"')=1
   ```

5. Searching for terms in the same sentence or paragraph.

   You can search for several terms that occur in one sentence or paragraph, such as:

   ```
   CONTAINS(commenthandle, '"compress" \
   IN SAME SENTENCE AS "DB2"') = 1

   CONTAINS(commenthandle, '"compress" \
   IN SAME PARAGRAPH AS "DB2" AND "compiler"') = 1
   ```

6. Searching for synonyms of terms.

   If you use a linguistic or dual index, you can search for a term and its synonyms. The DB2 Text Extender provides a dictionary that lists synonyms for many terms. The default dictionary is always US_English. To search for synonyms of a term:

   ```
   CONTAINS(commenthandle, 'SYNONYM FORM OF UK_ENGLISH "book"')=1
   ```

   You do not need to specify "US_ENGLISH" if you want to use the US_English dictionary.

6.7 Creating Applications

The DB2 Text Extender provides API functions to enable applications to use the extender utilities. Some of these API functions require a database connection handle, obtained from the SQLConnect() function, as an input parameter. This restricts the use of these API functions only to DB2 CLI applications. They cannot be used in Embedded SQL applications. But API functions can be used in mixed applications which use both CLI and Embedded SQL. Refer to the *Call Level Interface Guide and Reference - for common servers* on how to write mixed applications.

The APIs provided by the DB2 Text Extender are:

1. API that enables a search function:
   - DesGetSearchResultTable()

     This function is used to search through text documents using a search argument. It can find the text handle data, rank and number of matches, and write this information to the result table (see 6.7.1, “Starting the Browse” on page 135 for details about the result table).
2. APIs that enable browse functions:
   - DesGetBrowseInfo()

     This function is needed to get a pointer to browse information. This pointer is used in the DesStartBrowseSession() function to highlight the found terms.
   - DesStartBrowseSession()

     The purpose of this function is to open a browse session for browsing text documents. Users will be prompted for a user ID and password login for authorization.
   - DesBrowseDocument()

     If you are using the Text Extender browser to display a text document, you will need to call this function which will start the DB2 Text Extender browser. The search terms will be highlighted.
   - DesOpenDocument()

     This function prepares the text document that corresponds to the handle returned from the DesStartBrowseSession() function. It will highlight the information and return a document handle used for calling the DesGetMatches().
   - DesGetMatches()

This function is used to obtain a pointer to a data stream containing the highlighted information for the text document described by a document handle.

- `DesCloseDocument()`

  We use this function to close a text document that was opened by the `DesOpenDocument()` function and to release the storage used during the return of document text and the highlighted information.

- `DesEndBrowseSession()`

  This function is used to end a browse session and to release the storage allocated for the browse session.

- `DesFreeBrowseInfo()`

  The purpose of this function is to free the storage allocated for the browse information by the `DesGetBrowseInfo()`.

For further information about the DB2 Text Extender's API functions, such as the arguments and their data types, the syntax, and so on, refer to the *DATABASE 2 Text Extender: Administration and Programming*.

When you create an application using the DB2 Text Extender's API functions, you need to do the following:

1. Include the des_ext.h file in the program. This file is provided by the DB2 Text Extender and is located in the /include subdirectory of your /home/$DB2TX_INSTOWNER/db2tx directory or in the Text Extender installation directory (/usr/lpp/db2tx_01_01_0000/include).
2. Compile your application either using the C or C + + language.

   ```
   xlC -I/$DB2TX_INSTOWNER/include -I/$DB2TX_INSTOWNER/db2tx/include $1.c
   ```

   where

`$DB2TX_INSTOWNER`	The home directory of the DB2 Text Extender instance owner
`$1`	The name of the application program

3. Link the library libdescl.a to your application. This library is located in the /lib subdirectory of the DB2 Text Extender installation directory.

   ```
   xlC -o $1 $1.o -ldb2 -L/home/db2/sqllib/lib \
   -ldescl -L/usr/lpp/db2tx_01_01_0000/lib -lc
   ```

Basically, there are three parts of an application that will browse an extender document. These are:

1. Starting the browse
2. Browsing documents
3. Ending the browse

6.7.1 Starting the Browse

To start a browse, you need to perform the following steps:

1. Get the browse information

   To get the browse information, we can use one of the following functions:

   a. Get Browse Information

The function used is DesGetBrowseInfo(). This function returns a pointer to the browser information which consists of a list of all the terms to be highlighted. The browser information is needed to start a browse session.

b. Get Search Result Table

The DesGetSearchResultTable() function is used by an application program to establish a connection to the database and to search for terms in a particular text column. The result of the search contains the text handles in each document found. This data is stored in a table named the Result Table which must be created before calling the function by using the following structure:

- TEXTHANDLE with DB2TEXTH type
- RANK with DOUBLE type
- MATCHES with INTEGER type

Any user who will run an application using this function must have at least the DELETE, INSERT, and SELECT privilege on the result table used by the application. This is because each time you call the function, the following steps are performed:

1) Any existing rows in the result table are deleted.
2) The results of running DesGetSearchResultTable() function are inserted into the result table.
3) The application performs SELECT tasks on the result table.

All these steps are done using the DB2 user ID and password authorization provided in the DesGetSearchResultTable() input function.

Using the DesGetSearchResultTable() function in an application may be faster than using the UDFs provided by the DB2 Text Extender when working in a text-only query. This is because the API function goes directly to the server to get the rank and number of matches, and it loops only for the number of matching documents found. The DesGetSearchResultTable() function can be used only on base tables. They are tables created using the CREATE TABLE statement.

2. Start a Browse Session

The application uses the DesStartBrowseSession() function to start a browse session. This function gets a pointer to the browse information either from the DesGetBrowseInfo() or the DesGetSearchResultTable() function. And it returns a browse session handle that is required by the other browse functions.

6.7.2 Browsing Documents

To browse the documents found in the search, you can use the DB2 Text Extender Browser or use your own browser.

6.7.2.1 Using the DB2 Text Extender Browser

Figure 78 on page 137 shows the order of functions used in an application that browses text using DB2 Text Extender's browser.

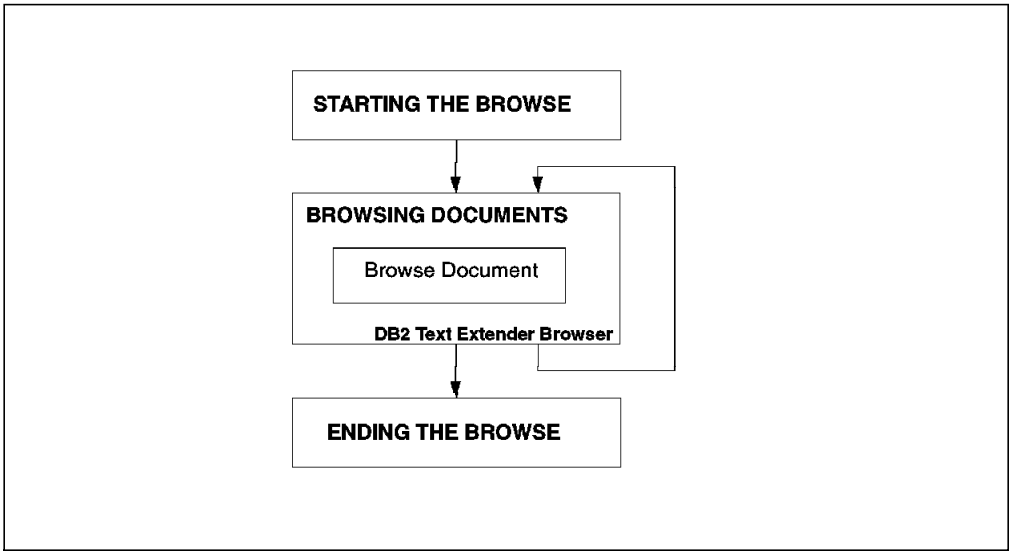


Figure 78. API Functions Using the DB2 Text Extender Browser

In this step, the application uses the DB2 Text Extender browser by calling the `DesBrowseDocument()` function. The browser will display the text document specified by the text handle and present the search terms highlighted (refer to 6.8.1, “Browse Application Using the DB2 Text Extender Browser” on page 139). The application can call this function several times for different text documents within one browse session.

6.7.2.2 Using Your Own Browser

Figure 79 shows the steps to use your own browser in a browse application.

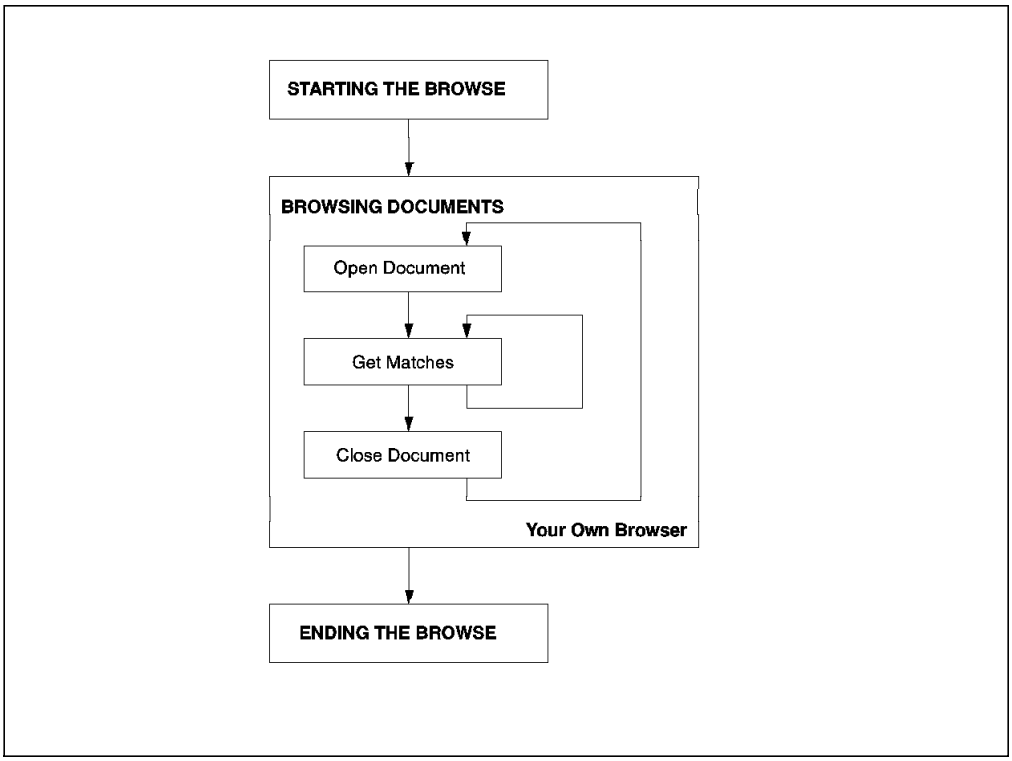


Figure 79. API Functions Using Your Own Browser

The following steps are needed when you want to use your own browser:

1. Open a Document

   We use the DesOpenDocument() function to open a document. The purpose of this function is to prepare the text document corresponding with the text handle, get the document text and highlight the information. One of the inputs for this function is a pointer to the browse session handle resulting from starting a browse session. The output of this function is a document handle that will be used in the DesGetMatches() function.

2. Get Matches

   The DesGetMatches() function will return a pointer to the highlighted information for the text document in the form of a data stream. The application then parses the data stream and processes it using the user's own browser.

   The data stream returned from DesGetMatches() is only a portion of the stream which indicates the length of the portion in the structure. The DesGetMatches() function will be called repeatedly until the entire text document is obtained. At that time, an indicator will be returned.

3. Close a Document

   The text document, which was previously opened by the DesGetMatches() function, is closed by calling the DesCloseDocument() function. The allocated storage used during the retrieval of the text document, and the highlighted information, is also freed.

6.7.3 Ending the Browse

The following steps are used to end a browse:

1. End a Browse Session

   To end a browse session, we use the DesEndBrowseSession() function. This function will end a browse session that was started by DesStartBrowseSession() and release the storage allocated for this particular browse session.

2. Free the Browse Information

   The storage allocated for the browse information is released by calling the DesFreeBrowseInfo() function.

6.7.4 Checking the Browser

Before running your applications using the DB2 Text Extender Browser or your own browser, you should verify the following:

- Check that you have the appropriate privileges to use the browser, such as having execute permission in the browser files. For the DB2 Text Extender Browser, verify <inst_name>/db2tx/bin files.
- If you are getting the message:

  ```
  Message catalog files not found
  ```

  Verify the environment variable LANG and the executable permission in the <inst_name>/db2tx/msg files.

- If you are getting the error message:

  ```
  DES0369N The browse process was started but did not respond
  in an appropriate time.
  ```

Verify the system variable DISPLAY that must be set up to use the graphic mode. For example:

```
export DISPLAY=tx8.itsc.austin.ibm.com:0
```

6.7.5 Return Codes

As in DB2 Call Level Interface functions, the DB2 Text Extender may also return a code that tells whether an error occurred while processing the function or not. Refer to the *DATABASE 2 Text Extender: Administration and Programming* manual for detailed information on each return code.

Figure 80 shows an example of how to use the return codes in an application.

```
...
DESrc = DesGetSearchResultTable
                 ( hdbc, pTableName, TableNameLength, pColumnName,
                   ColumnNameLength, srchArg, pColumnName,
                   ArgumentLength, pResultTableName, ResultNameLength,
                   pSchemaName, SchemaNameLength, SearchOption,
                   BrowseOption, &BrowseInfo, &ErrorMessage);

switch (DESrc) {
        case RC_SUCCESS:
             break;
        case RC_SE_DICTIONARY_NOT_FOUND:
             WARNrc = RC_SE_DICTIONARY_NOT_FOUND;
             break;
        case RC_SE_STOPWORD_IGNORED:
             WARNrc = RC_SE_STOPWORD_IGNORED;
             break;
        case RC_SE_CONFLICT_WITH_INDEX_TYPE:
             WARNrc = RC_SE_CONFLICT_WITH_INDEX_TYPE;
             break;
        case RC_SE_NO_DATA:
                terminateDB ( henv, hdbc );
                return(DESrc);
        default:
        }
...
```

Figure 80. Using Return Codes in an Application

6.8 Examples

The following examples will show you how to write browse applications using the DB2 Text Extender functions to search in LOB columns.

6.8.1 Browse Application Using the DB2 Text Extender Browser

The Figure 81 on page 140 contains a fragment of the sample1.c application. This application uses the DB2 Text Extender Browser to present the information found in the sample table. The complete code of sample1.c is included in A.3, “Text Extender Example (sample1.c)” on page 148.

```
...

/* Call DesGetSearchResultTable                                    1
*-----------------------------*/
DESrc = DesGetSearchResultTable
            ( hdbc, pTableName, TableNameLength, pColumnName,
              ColumnNameLength, srchArg, ArgumentLength,
              pResultTableName, ResultNameLength, pSchemaName,
              SchemaNameLength, SearchOption, BrowseOption,
             &BrowseInfo, &ErrorMessage);

/* Use the Return Code Messaging                                   2
*-------------------------------*/
...

if (BrowseOption == DES_NO_BROWSE) {                              3
       DESrc = terminateDB ( henv, hdbc );
       if (WARNrc != RC_SUCCESS)
           return (WARNrc);
       return(DESrc);
    }

/* Start a browse session                                          4
/*----------------------*/
   DESrc = DesStartBrowseSession ( BrowseInfo, (char *) pUserId, DES_NTS,
                                    (char *) pPassword, DES_NTS,
                                   &BrowseSession, &ErrorMessage);
      if (DESrc != RC_SUCCESS) {
          printf ( "Error message: %s\n", ErrorMessage);
          DesFreeBrowseInfo (BrowseInfo );
          return(DESrc);
       }

/* Browse a document                                               5
/*-----------------*/
    DESrc = DesBrowseDocument( BrowseSession, (SQLCHAR *) pHandle,
                              HandleLength, &WindowHandle, &ErrorMessage);
    if (DESrc != RC_SUCCESS) {
           printf ( "Error message: %s\n", ErrorMessage);
           if (RC_SUCCESS != (rc =  DesEndBrowseSession (BrowseSession)))
              printf (" DesEndBrowseSession returned %d\n",rc);
           DesFreeBrowseInfo (BrowseInfo );
           return(DESrc);
        }

/* End the browse session                                          6
/*----------------------*/
    DESrc =  DesEndBrowseSession (BrowseSession);
    if (DESrc != RC_SUCCESS) {
          DesFreeBrowseInfo (BrowseInfo );
          return(DESrc);
    }
    DESrc = DesFreeBrowseInfo (BrowseInfo );
    if (DESrc != RC_SUCCESS) {
          return(DESrc);
     }
   if (WARNrc != RC_SUCCESS)
          return (WARNrc);
          return(DESrc);
    }
...
```

Figure 81. Using the DB2 Text Extender Browser: sample1.c

1 First step is calling the DesGetSearchResultTable() function to establish the connection and getting the pointer to the browse information.

2 You can process the return code as shown in the example in Figure 80 on page 139.

3 Ends the connection if the DES_NO_BROWSE option is chosen. At this point, the result table has been updated with the result of the search.

4 Start the browse session by calling the DesStartBrowseSession() function. The first parameter, BrowseInfo, is taken from the DesGetSearchResultTable() pointer. If the browser is not responding, an error handling routine frees the browse session (refer to 6.7.4, "Checking the Browser" on page 138 for possible causes for why the browser may not be responding).

5 The DB2 Text Extender Browser uses the information of the browse session started to browse the selected document.

6 Ends the browse session and frees the resources allocated for this particular session.

Figure 82 is the interface to input parameters to the browse application, and the result is presented by the DB2 Text Extender Browser in a window like that shown in Figure 83 on page 142.

```
Enter the database name
celdial

Enter the user ID
db2

Enter the password
db2

Enter the table name
SAMPLE

Enter the handle column name
COMMENTHANDLE

Enter the result table name
RESULT

Enter the schema name of the result table
DB2

Enter the search option
DES_RANKANDMATCH

Enter the browse option
DES_BROWSE

Enter the search argument
"compress"
```

Figure 82. User Interface to Input Parameters Running sample1.c

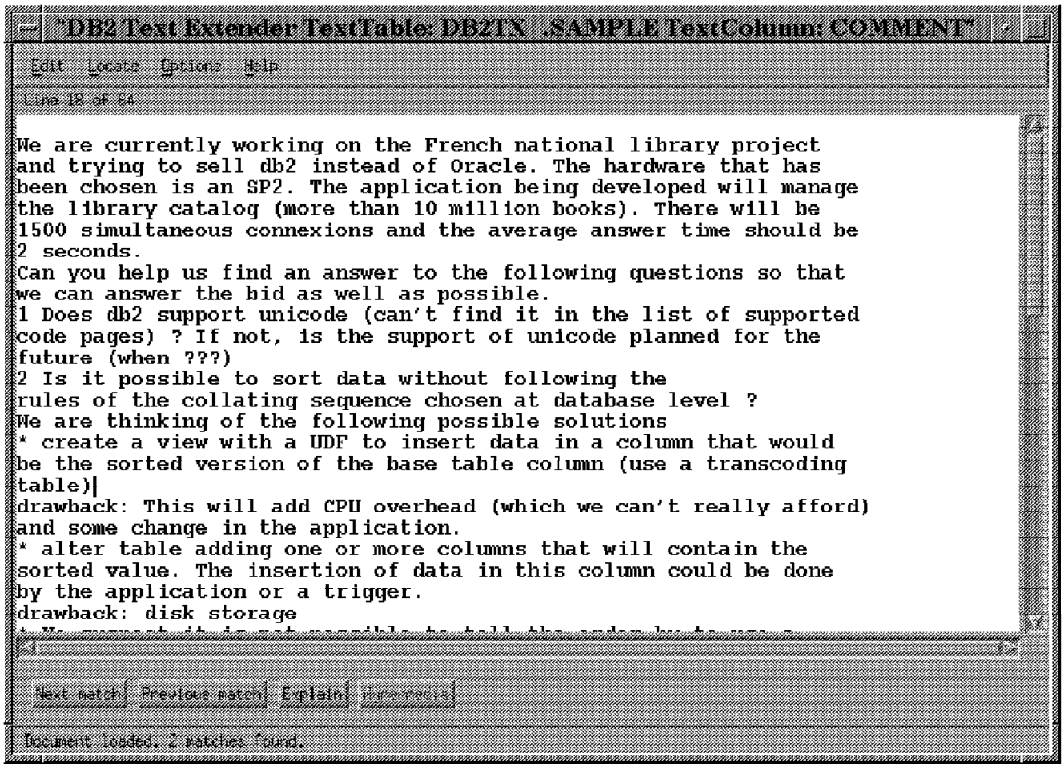


Figure 83. Browsing a Document Using the DB2 Text Extender Browser

To check the information stored in the result table, you can perform the following SELECT from the CLP:

```
SELECT * FROM db2.result
```

The result should be similar to the output shown in Figure 84.

```
TEXTHANDLE                          RANK                    MATCHES
----------------------------------- ----------------------- -----------
x'4131000173F712E2CFA10000020 . . .    +1.90977560414150E-001           2
x'4131000173F712E2DBB . . . . . .    +2.10666504772403E-001           2

  2 record(s) selected.
```

Figure 84. Search Result Stored in the Result Table

Appendix A. Sample Applications

This appendix contains a complete listing of the sample applications used within this book.

A.1 CLI Example (listcol.c)

```
/************************************************************************
**
** Source File Name = listcol.c
**
************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "sqlcli.h"
#include "sqlcli1.h"

#define MAX_UID_LENGTH    15
#define MAX_PWD_LENGTH    15

/************************************************************************
** main
************************************************************************/
int
main( int argc, char * argv[] )
{
    SQLCHAR         dbname[SQL_MAX_DSN_LENGTH + 1];
    SQLCHAR         uid[MAX_UID_LENGTH + 1];
    SQLCHAR         pwd[MAX_PWD_LENGTH + 1];

    SQLHENV         henv;
    SQLHDBC         hdbc;
    SQLHSTMT        hstmt;
    SQLRETURN       rc;

    struct {  SQLINTEGER   ind;
              SQLCHAR      s[129];
           } table_schem, table_name, column_name, type_name;

    SQLINTEGER      length_ind;
    SQLINTEGER      length;

    SQLCHAR         buffer[SQL_MAX_MESSAGE_LENGTH + 1];
    SQLCHAR         sqlstate[SQL_SQLSTATE_SIZE + 1];
    SQLINTEGER      sqlcode;
    SQLSMALLINT     elength;

    SQLAllocEnv(&henv);

    printf("Enter Database Name : ");
    gets((char *) dbname);
    printf("Enter User Name     : ");
    gets((char *) uid);
    printf("Enter Password for %s : ", uid);
    gets((char *) pwd);
```

```
SQLAllocConnect(henv, &hdbc);

rc = SQLConnect(hdbc, dbname, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);

if (rc != SQL_SUCCESS) {
   printf("---ERROR while connecting to database : %s ---\n", dbname)
   while (SQLError(henv, hdbc, hstmt, sqlstate, &sqlcode, buffer,
          SQL_MAX_MESSAGE_LENGTH + 1, &elength) == SQL_SUCCESS) {
          printf("         SQLSTATE: %s\n", sqlstate);
          printf("Native Error Code: %ld\n", sqlcode);
          printf("%s \n", buffer);
   };
   return (SQL_ERROR);
} else {
   printf("\n--- Connected to database : %s ---\n\n", dbname);
};

SQLAllocStmt(hdbc, &hstmt);

printf("Enter Table Schema Name Search Pattern (in uppercase) :\n");
gets((char *)table_schem.s);
printf("Enter Table Name Search Pattern (in uppercase) :\n");
gets((char *)table_name.s);

SQLColumns(hstmt, NULL, 0, table_schem.s, SQL_NTS,
               table_name.s, SQL_NTS, (SQLCHAR *)"%", SQL_NTS);

SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER) table_schem.s, 129,
               &table_schem.ind);

SQLBindCol(hstmt, 3, SQL_C_CHAR, (SQLPOINTER) table_name.s, 129,
               &table_name.ind);

SQLBindCol(hstmt, 4, SQL_C_CHAR, (SQLPOINTER) column_name.s, 129,
               &column_name.ind);

SQLBindCol(hstmt, 6, SQL_C_CHAR, (SQLPOINTER) type_name.s, 129,
               &type_name.ind);

SQLBindCol(hstmt, 7, SQL_C_LONG, (SQLPOINTER) & length,
               sizeof(length), &length_ind);

/* Fetch each row, and display */

printf("   \n");
printf("   \n");
printf("TABLE          SCHEMA   COLUMN            TYPE
       LENGTH\n");
printf("-------------- -------- ----------------- ------------------
       ------\n");

while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
    printf("%-14.14s %-8.8s %-17.17s %-18.18s %-6.6ld\n",
    table_name.s, table_schem.s, column_name.s, type_name.s, length);
}                          /* endwhile */

SQLFreeStmt(hstmt, SQL_DROP);
```

```
    SQLTransact(henv, hdbc, SQL_COMMIT);


    printf("Disconnecting .....\n");
    SQLDisconnect(hdbc);

    SQLFreeConnect(hdbc);

    SQLFreeEnv(henv);

    return (SQL_SUCCESS);
}                                /* end main */
```

A.2 CLI LOB Example (lookres.c)

```
/************************************************************************
 **
 ** Source File Name = lookres.c  %I%
 **
 ************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "sqlcli.h"
#include "sqlcli1.h"
#include "samputil.h"

#define MAX_STMT_LEN 255
#define ROWSET_SIZE  10

SQLCHAR   server[SQL_MAX_DSN_LENGTH + 1];
SQLCHAR   uid[MAX_UID_LENGTH + 1];
SQLCHAR   pwd[MAX_PWD_LENGTH + 1];

/************************************************************************
** main
************************************************************************/
int
main( int argc, char * argv[] ) {
    SQLHENV     henv;
    SQLHDBC     hdbc;
    SQLHSTMT    hstmt, lhstmt;
    SQLRETURN   rc;
    SQLCHAR     stmt1[] =
                "SELECT employee.empno, firstnme || lastname as name "
                 "FROM employee, emp_resume "
                 "WHERE employee.empno = emp_resume.empno "
                       "AND resume_format = 'ascii'";
    SQLCHAR     stmt2[] =
                "SELECT resume FROM emp_resume "
                 "WHERE empno = ?  AND resume_format = 'ascii'";
    SQLCHAR     stmt3[] = "FREE LOCATOR ?";
    struct {
              SQLINTEGER      ind;
              SQLCHAR         s[30];
           } Name;
    struct {
              SQLINTEGER      ind;
```

```
          SQLCHAR         s[7];
       } Empno;
SQLINTEGER      ClobLoc1;
SQLINTEGER      pcbValue;
SQLINTEGER      SLength;
SQLUINTEGER     Pos1;
SQLINTEGER      OutLength, Ind;
SQLCHAR       * buffer;

INIT_UID_PWD;

rc = SQLAllocEnv(&henv);
if (rc != SQL_SUCCESS)
    return (terminate(henv, rc));
rc = DBconnect(henv, &hdbc);
if (rc != SQL_SUCCESS)
    return (terminate(henv, rc));

rc = SQLAllocStmt(hdbc, &hstmt);
CHECK_DBC(hdbc, rc);

rc = SQLExecDirect(hstmt, stmt1, SQL_NTS);
CHECK_STMT(hstmt, rc);

rc = SQLBindCol(hstmt, 1,SQL_C_CHAR, Empno.s, 7,&Empno.ind);
CHECK_STMT(hstmt, rc);
rc = SQLBindCol(hstmt, 2,SQL_C_CHAR, Name.s, 30,&Name.ind);
CHECK_STMT(hstmt, rc);

printf("\nEmpno   Name \n");
printf("------- --------------------\n");
while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
    printf("%-6s %-30s \n", Empno.s, Name.s);
}
if (rc != SQL_NO_DATA_FOUND)
    check_error(henv, hdbc, hstmt, rc, __LINE__, __FILE__);

rc = SQLFreeStmt(hstmt, SQL_CLOSE); CHECK_STMT(hstmt, rc);
rc = SQLFreeStmt(hstmt, SQL_UNBIND); CHECK_STMT(hstmt, rc);
rc = SQLFreeStmt(hstmt, SQL_RESET_PARAMS); CHECK_STMT(hstmt, rc);

rc = SQLSetParam(hstmt, 1, SQL_C_CHAR, SQL_CHAR, 7,
                 0, Empno.s, &Empno.ind);
CHECK_STMT(hstmt, rc);

printf("\n>Enter an employee number:\n");
gets((char *)Empno.s);

rc = SQLExecDirect(hstmt, stmt2, SQL_NTS);
CHECK_STMT(hstmt, rc);

rc = SQLBindCol(hstmt, 1, SQL_C_CLOB_LOCATOR, &ClobLoc1, 0,
                 &pcbValue);
CHECK_STMT(hstmt, rc);

rc = SQLFetch(hstmt);
CHECK_STMT(hstmt, rc);

rc = SQLAllocStmt(hdbc, &lhstmt);
```

```
CHECK_DBC(hdbc, rc);

rc = SQLGetLength(lhstmt, SQL_C_CLOB_LOCATOR, ClobLoc1,
                  &SLength, &Ind);
CHECK_STMT(lhstmt, rc);

rc = SQLGetPosition(lhstmt, SQL_C_CLOB_LOCATOR, ClobLoc1, 0,
                (SQLCHAR *)"Interests", 9, 1, &Pos1, &Ind);
CHECK_STMT(lhstmt, rc);

rc = SQLFreeStmt(lhstmt, SQL_CLOSE); CHECK_STMT(lhstmt, rc);

buffer = (SQLCHAR *)malloc(SLength - Pos1 + 1);

rc = SQLGetSubString(lhstmt, SQL_C_CLOB_LOCATOR, ClobLoc1, Pos1,
     SLength - Pos1, SQL_C_CHAR, buffer, SLength - Pos1 +1,
     &OutLength, &Ind);
CHECK_STMT(lhstmt, rc);

printf("\nEmployee #: %s\n %s\n", Empno.s, buffer);

rc = SQLFreeStmt(hstmt, SQL_UNBIND);       CHECK_STMT(hstmt, rc);
rc = SQLFreeStmt(hstmt, SQL_RESET_PARAMS); CHECK_STMT(hstmt, rc);
rc = SQLFreeStmt(hstmt, SQL_CLOSE);        CHECK_STMT(hstmt, rc);

rc = SQLSetParam(hstmt, 1, SQL_C_CLOB_LOCATOR,
                 SQL_CLOB_LOCATOR, 0, 0, &ClobLoc1, NULL);
CHECK_STMT(hstmt, rc);

rc = SQLExecDirect(hstmt, stmt3, SQL_NTS);
CHECK_STMT(hstmt, rc);

rc = SQLFreeStmt(hstmt, SQL_DROP);  CHECK_STMT(hstmt, rc);
rc = SQLFreeStmt(lhstmt, SQL_DROP); CHECK_STMT(lhstmt, rc);

rc = SQLTransact(henv, hdbc, SQL_COMMIT);
CHECK_DBC(hdbc, rc);

printf(">Disconnecting .....\n");
rc = SQLDisconnect(hdbc);
CHECK_DBC(hdbc, rc);

rc = SQLFreeConnect(hdbc);
CHECK_DBC(hdbc, rc);

rc = SQLFreeEnv(henv);
if (rc != SQL_SUCCESS)
    return (terminate(henv, rc));

return (SQL_SUCCESS);
}
```

A.3 Text Extender Example (sample1.c)

```
/************************************************************************
**
** Source File Name = sample1.c  %I%
**
************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <des_ext.h>


int get_text_handle ( SQLHENV           henv,
                      SQLHDBC           hdbc,
                      char*             pHandle,
                      DESSMALLINT*      pHandleLength,
                      char *            pSchemaName,
                      DESSMALLINT       SchemaNameLength,
                      char *            pResultTableName,
                      DESSMALLINT       ResultNameLength,
                      DESSEARCHOPTION   SearchOption);

int initialize      ( SQLHENV          *henv,
                      SQLHDBC          *hdbc,
                      SQLCHAR          *pdbName,
                      SQLCHAR          *pUserId,
                      SQLCHAR          *pPassword);

void getInputData   ( char             *pTableName,
                      DESSMALLINT      *pTableNameLength,
                      char             *pColumnName,
                      DESSMALLINT      *pColumnNameLength,
                      char             *srchArg,
                      DESSMALLINT      *pArgumentLength,
                      char             *pResultTableName,
                      DESSMALLINT      *pResultNameLength,
                      char             *pSchemaName,
                      DESSMALLINT      *pSchemaNameLength,
                      DESSEARCHOPTION  *pSearchOption,
                      DESBROWSEOPTION  *pBrowseOption);

void printSqlErrMsg ( SQLHENV           henv,
                      SQLHDBC           hdbc,
                      SQLHSTMT          hstmt);

int terminateDB     ( SQLHENV           henv,
                      SQLHDBC           hdbc );

int
main (int args, char *argv[])
 {
    /*-----------------------------------------------------------------------*/
    /* Declarations                                                          */
    /*-----------------------------------------------------------------------*/
    DESRETURN        DESrc  = RC_SUCCESS;
    SQLRETURN        SQLrc  = RC_SUCCESS;
    SQLRETURN        WARNrc = RC_SUCCESS;
```

```
SQLRETURN        rc     = RC_SUCCESS;

SQLHENV          henv   = SQL_NULL_HENV;
SQLHDBC          hdbc   = SQL_NULL_HDBC;

SQLCHAR          dbName    [18+1];
SQLCHAR          pUserId   [ 8+1];
SQLCHAR          pPassword [18+1];

DESMESSAGE       ErrorMessage;

char             pTableName[18+1];
DESSMALLINT      TableNameLength = DES_NTS;
char             pColumnName[18 +1];
DESSMALLINT      ColumnNameLength = DES_NTS;
char             srchArg  [1000];
DESSMALLINT      ArgumentLength = DES_NTS;
char             pResultTableName[18+1];
DESSMALLINT      ResultNameLength;
char             pSchemaName [8 +1];
DESSMALLINT      SchemaNameLength;
DESSEARCHOPTION  SearchOption;
DESBROWSEOPTION  BrowseOption;

DESBROWSEINFO    BrowseInfo ;
DESBROWSESESSION BrowseSession;
char             pHandle[256];
DESSMALLINT      HandleLength  ;

DESWINDOWHANDLE  WindowHandle;

/*-------------------------------------------------------------------------*/
/* Program logic                                                           */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/* Get input data for connection to the database                           */
/*-------------------------------------------------------------------------*/
printf("\nEnter the database name \n");
scanf("%s",dbName);

printf("\nEnter the user ID \n");
scanf("%s", pUserId);

printf("\nEnter the password\n");
scanf("%s", pPassword);

/*-------------------------------------------------------------------------*/
/* Initialize the database                                                 */
/*-------------------------------------------------------------------------*/
DESrc = initialize ( &henv, &hdbc, dbName, pUserId, pPassword);

  if (DESrc != RC_SUCCESS)  {
      printf(" initialize      ..... %d \n", SQLrc );
      return (DESrc);
   }

/*-------------------------------------------------------------------------*/
/* Get input for DesGetSearchResultTable                                   */
```

```
/*--------------------------------------------------------------------*/
getInputData  ( pTableName, &TableNameLength,
                pColumnName, &ColumnNameLength,
                srchArg, &ArgumentLength,
                pResultTableName, &ResultNameLength,
                pSchemaName, &SchemaNameLength,
                &SearchOption, &BrowseOption);

/*--------------------------------------------------------------------*/
/* Call DesGetSearchResultTable                                       */
/*--------------------------------------------------------------------*/
DESrc = DesGetSearchResultTable
              ( hdbc,                  /* in  CLI connection handle     */
                pTableName,            /* in  name of target table      */
                TableNameLength,       /* in  length of table name      */
                pColumnName,           /* in  name of target column     */
                ColumnNameLength,      /* in  length of column name     */
                srchArg,               /* in  search argument           */
                ArgumentLength,        /* in  search argument length    */
                pResultTableName,      /* in  name of result table      */
                ResultNameLength,      /* in  length of result table    */
                pSchemaName,           /* in  schema                    */
                SchemaNameLength,      /* in  schema length             */
                SearchOption,          /* in  search option             */
                BrowseOption,          /* in  browse option             */
               &BrowseInfo,            /* out browsing information      */
               &ErrorMessage);

switch (DESrc) {
    case RC_SUCCESS:
          break;
    case RC_SE_DICTIONARY_NOT_FOUND:
          WARNrc = RC_SE_DICTIONARY_NOT_FOUND;
          break;
    case RC_SE_STOPWORD_IGNORED:
          WARNrc = RC_SE_STOPWORD_IGNORED;
          break;
    case RC_SE_CONFLICT_WITH_INDEX_TYPE:
          WARNrc = RC_SE_CONFLICT_WITH_INDEX_TYPE;
          break;
    case RC_SE_NO_DATA:
             terminateDB ( henv, hdbc );
             return(DESrc);
    default: /* an error occurred                                      */
            printf ( "Error message: %s\n", ErrorMessage);
            terminateDB ( henv, hdbc );
            return(DESrc);
   }

if (BrowseOption == DES_NO_BROWSE) {
   DESrc = terminateDB ( henv, hdbc );
   if (WARNrc != RC_SUCCESS)
       return (WARNrc);
   return(DESrc);
}

/*--------------------------------------------------------------------*/
/* Get a handle from the result table                                 */
/* If the search option is DES_RANK, get the one with the highest rank   */
```

```
/*-------------------------------------------------------------------------*/
DESrc = get_text_handle ( henv, hdbc,
                          pHandle, &HandleLength,
                          pSchemaName, SchemaNameLength,
                          pResultTableName, ResultNameLength,SearchOption);

  if (DESrc != RC_SUCCESS) {
      printf ( "\nget_text_handle: DESrc = %d\n",DESrc);
      terminateDB ( henv, hdbc );
      return(DESrc);
   }

DESrc = terminateDB (henv, hdbc);

/*-------------------------------------------------------------------------*/
/* Start a browse session                                                  */
/*-------------------------------------------------------------------------*/
DESrc = DesStartBrowseSession ( BrowseInfo,
                                (char *) pUserId, DES_NTS,
                                (char *) pPassword, DES_NTS,
                               &BrowseSession,
                               &ErrorMessage);

  if (DESrc != RC_SUCCESS) {
      printf ( "Error message: %s\n", ErrorMessage);
      DesFreeBrowseInfo (BrowseInfo );
      return(DESrc);
   }

/*-------------------------------------------------------------------------*/
/* Browse a document                                                       */
/*-------------------------------------------------------------------------*/
DESrc = DesBrowseDocument( BrowseSession,
                           (SQLCHAR *) pHandle,
                           HandleLength,
                          &WindowHandle,
                          &ErrorMessage);

    if (DESrc != RC_SUCCESS) {
       printf ( "Error message: %s\n", ErrorMessage);
       if (RC_SUCCESS != (rc =  DesEndBrowseSession (BrowseSession)))
          printf (" DesEndBrowseSession returned %d\n",rc);
       DesFreeBrowseInfo (BrowseInfo );
       return(DESrc);
    }

/*-------------------------------------------------------------------------*/
/* End the browse session                                                  */
/*-------------------------------------------------------------------------*/
DESrc =  DesEndBrowseSession (BrowseSession);

  if (DESrc != RC_SUCCESS) {
      DesFreeBrowseInfo (BrowseInfo );
      return(DESrc);
   }

DESrc = DesFreeBrowseInfo (BrowseInfo );
  if (DESrc != RC_SUCCESS) {
      return(DESrc);
```

```
      }


   if (WARNrc != RC_SUCCESS)
      return (WARNrc);
   return(DESrc);
}

/*============================================================================
 *      Get the text handle having highest rank or highest match
 *      (if applicable) from the previous text search
 *============================================================================*/
int get_text_handle ( SQLHENV         henv,
                      SQLHDBC         hdbc,
                      char*           pHandle,
                      DESSMALLINT*    pHandleLength,
                      char *          pSchemaName,
                      DESSMALLINT     SchemaNameLength,
                      char *          pTableName,
                      DESSMALLINT     TableNameLength,
                      DESSEARCHOPTION SearchOption)
{

   SQLRETURN        SQLrc  = RC_SUCCESS;
   SQLHSTMT         hstmt;
   SQLINTEGER       docHdlLenTemp;
   SQLINTEGER       lengthIn;
   SQLINTEGER       lengthOut;
   unsigned short   rank;
   unsigned long    matchinfo;

   char             t [ 18 +1];
   char             s [  8 +1];

   char SqlStmtMsk[] ="SELECT texthandle from \"%s\".\"%s\" ";
   char SqlStmtMskMI[]="SELECT texthandle,"
                       " rank from \"%s\".\"%s\" order by 2";

   char SqlStmt [sizeof(SqlStmtMskMI)+8+18 +1];

   /*-------------------------------------------------------------------------*/
   /* Build the SQl statement                                                 */
   /*-------------------------------------------------------------------------*/
   if (SchemaNameLength == DES_NTS)
      strcpy (s ,pSchemaName );
   else  {
      memset (s, '\0', 8 +1);
      memcpy (s ,pSchemaName ,SchemaNameLength );
   }

   if (TableNameLength == DES_NTS)
      strcpy (t ,pTableName );
   else  {
      memset (t, '\0', 18 +1);
      memcpy (t ,pTableName ,TableNameLength );
   }


   if ( SearchOption == DES_TEXTHANDLEONLY || SearchOption == DES_MATCH )
```

```
    sprintf ( (char *) SqlStmt, (char *)SqlStmtMsk ,s,t);
  else
    sprintf ( (char *) SqlStmt, (char *)SqlStmtMskMI ,s,t);

  /*-------------------------------------------------------------------------*/
  /* Alloc the statement and execute it                                      */
  /*-------------------------------------------------------------------------*/
  SQLrc = SQLAllocStmt (hdbc, &hstmt) ;
    if (SQLrc != RC_SUCCESS) {
        printSqlErrMsg (henv, hdbc, SQL_NULL_HSTMT);
        return(SQLrc);
     }

  SQLrc = SQLExecDirect (hstmt, (SQLCHAR *) SqlStmt, SQL_NTS);
    if (SQLrc != RC_SUCCESS) {
        printSqlErrMsg (henv, hdbc, hstmt);
        SQLFreeStmt(hstmt,SQL_DROP);
        return(SQLrc);
     }

  /*-------------------------------------------------------------------------*/
  /* Bind the column                                                         */
  /*-------------------------------------------------------------------------*/
  SQLrc = SQLBindCol (hstmt, 1, SQL_C_BINARY,
                     (SQLPOINTER)pHandle,
                      DES_MAX_TEXT_HANDLE_LENGTH+1,
                      &docHdlLenTemp);

    if (SQLrc != RC_SUCCESS) {
        printSqlErrMsg (henv, hdbc, hstmt);
        SQLFreeStmt(hstmt,SQL_DROP);
        return(SQLrc);
     }

if ( SearchOption == DES_RANK || SearchOption == DES_RANKANDMATCH) {
       lengthIn = sizeof (double);
       SQLrc = SQLBindCol (hstmt, 2, SQL_C_DOUBLE, &rank,
                                     lengthIn, &lengthOut);
         if (SQLrc != RC_SUCCESS) {
             printSqlErrMsg (henv, hdbc, hstmt);
             SQLFreeStmt(hstmt,SQL_DROP);
             return(SQLrc);
          }
       }

  SQLrc = SQLFetch (hstmt);
    if (SQLrc != RC_SUCCESS ) {
      if ( SQLrc != SQL_NO_DATA_FOUND)
        printSqlErrMsg (henv, hdbc, hstmt);

      SQLFreeStmt(hstmt,SQL_DROP);
      return(SQLrc);
     }

  SQLrc = SQLFreeStmt(hstmt,SQL_DROP);
    if (SQLrc != RC_SUCCESS) {
        printSqlErrMsg (henv, hdbc, hstmt);
        return(SQLrc);
     }
```

```
    *pHandleLength = docHdlLenTemp;
    return (RC_SUCCESS);
 }

/*============================================================================
 * Terminate the database
 *===========================================================================*/
int terminateDB ( SQLHENV         henv,
                  SQLHDBC         hdbc)
   {

    SQLRETURN     SQLrc = SQL_SUCCESS;

    /*-------------------------------------------------------------------------*/
    /* free all of DB                                                          */
    /*-------------------------------------------------------------------------*/
    if (SQL_SUCCESS != (SQLrc = SQLDisconnect ( hdbc)))
         printSqlErrMsg (henv, hdbc, SQL_NULL_HSTMT);

    if (SQL_SUCCESS != (SQLrc = SQLFreeConnect(hdbc)))
         printSqlErrMsg (henv, hdbc, SQL_NULL_HSTMT);

    if (SQL_SUCCESS != (SQLrc = SQLFreeEnv(henv)))
         printSqlErrMsg (henv, SQL_NULL_HDBC, SQL_NULL_HSTMT);

    return (SQLrc);
  }

/*============================================================================
 * Initialize
 *===========================================================================*/
int initialize  (  SQLHENV        *henv,
                   SQLHDBC        *hdbc,
                   SQLCHAR        *dbName,
                   SQLCHAR        *pUserId,
                   SQLCHAR        *pPassword )
  {
    SQLRETURN        SQLrc = RC_SUCCESS;

    SQLrc = SQLAllocEnv (henv);
      if (SQLrc != SQL_SUCCESS) {
          printf(" SQLAllocEnv     ..... %d \n", SQLrc );
          return (SQLrc);
       }

    SQLrc = SQLAllocConnect (*henv, hdbc);
      if (SQLrc == SQL_ERROR) {
          printSqlErrMsg (*henv, SQL_NULL_HDBC, SQL_NULL_HSTMT);
          terminateDB ( *henv, SQL_NULL_HDBC );
          return (SQLrc);
       }

    SQLrc = SQLConnect ( *hdbc, dbName, SQL_NTS, pUserId, SQL_NTS,
                            pPassword, SQL_NTS );

      if (SQLrc != SQL_SUCCESS) {
         printSqlErrMsg (*henv, *hdbc, SQL_NULL_HSTMT);
         terminateDB ( *henv, *hdbc );
```

```
        return (SQLrc);
      }

    return (SQLrc);
  }

/*============================================================================
 * Get input data
 *============================================================================*/
void getInputData  ( char            *pTableName,
                     DESSMALLINT     *pTableNameLength,
                     char            *pColumnName,
                     DESSMALLINT     *pColumnNameLength,
                     char            *srchArg,
                     DESSMALLINT     *pArgumentLength,
                     char            *pResultTableName,
                     DESSMALLINT     *pResultNameLength,
                     char            *pSchemaName,
                     DESSMALLINT     *pSchemaNameLength,
                     DESSEARCHOPTION *pSearchOption,
                     DESBROWSEOPTION *pBrowseOption)
  {

    char  option[30];
    int   j;

    printf("\nEnter the table name \n");
    scanf( "%s", pTableName);
    *pTableNameLength = DES_NTS;

    printf("\nEnter the handle column name \n");
    scanf( "%s", pColumnName);
    *pColumnNameLength = DES_NTS;

    printf("\nEnter the result table name \n");
    scanf( "%s", pResultTableName);
    *pResultNameLength = SQL_NTS;

    printf("\nEnter the schema name of the result table\n");
    scanf( "%s", pSchemaName);
    *pSchemaNameLength = SQL_NTS;

    printf("\nEnter the search option \n");
    scanf( "%s", option);
     for (j = 0 ; j < strlen(option) ; j++)
        option[j] = (char) toupper((int) option[j]);

        if (strcmp(option,"DES_TEXTHANDLEONLY")== 0)
            *pSearchOption = DES_TEXTHANDLEONLY;
        if (strcmp(option,"DES_RANK")== 0)
            *pSearchOption = DES_RANK;
        if (strcmp(option,"DES_MATCH")== 0)
            *pSearchOption = DES_MATCH;
        if (strcmp(option,"DES_RANKANDMATCH")== 0)
            *pSearchOption = DES_RANKANDMATCH;

    printf("\nEnter the browse option \n");
     scanf( "%s", option);
     for (j = 0 ; j < strlen(option) ; j++)
```

```
        option[j] = (char) toupper((int) option[j]);

      if ( strcmp(option, "DES_NO_BROWSE") == 0)
         *pBrowseOption = DES_NO_BROWSE;
      if ( strcmp(option, "DES_BROWSE") == 0)
         *pBrowseOption = DES_BROWSE;

   printf("\nEnter the search argument \n");
   scanf( "%s", srchArg);
   *pArgumentLength = DES_NTS;

 }

/*============================================================================
 * Print an SQL error message
 *============================================================================*/
void printSqlErrMsg ( SQLHENV         henv,
                      SQLHDBC         hdbc,
                      SQLHSTMT        hstmt)
  {
    SQLCHAR         SqlState [6];
    SQLINTEGER      NativeError;
    SQLCHAR         ErrorMsg [SQL_MAX_MESSAGE_LENGTH+1];
    SQLSMALLINT     ErrorMsgLen;

    SQLError ( henv, hdbc,hstmt,SqlState,
                      &NativeError , ErrorMsg , SQL_MAX_MESSAGE_LENGTH,
                      &ErrorMsgLen );

    printf("\nAn SQL error occurred\n");
    printf("SQL state    : %s \n", SqlState);
    printf("Native error : %d \n",NativeError);
    printf("Error message: %s \n",ErrorMsg);

    return;
   }
```

List of Abbreviations

APA	all points addressable
API	Application Programming Interface
BLOB	Binary Large Object
CAE	Client Application Enabler
CCSID	Coded Character Set Identifier
CGI	Common Gateway Interface
CLI	Command Line Interface
CLOB	Character Large Object
CLP	Command Line Processor
DARI	Database Application Remote Interface
DBMS	Database Management Service
DBCLOB	Double-Byte Character Large Obejct
DDCS	Distributed Database Connection Services
DRDA	Distributed Relational Database Architecture
DUOW	Distributed Unit of Work
GUI	Graphical User Interface
IBM	International Business Machines Corporation
ITSO	International Technical Support Organization
LOB	Large Object
OBDC	Outboard Device Controller
ODBC	Open Database Connectivity
PROFS	Professional Office System
SMIT	System Management Interface Tool
SQL	Structured Query Language
SQLCA	Structured Query Language Communications Area
SQLDA	Structured Query Language Data Area
SSL	Secured Sockets Layer
UDF	User-Defined Function
UDT	User-Defined Type

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

ITSO Redbook Evaluation

International Technical Support Organization
DATABASE 2 for AIX
Programming Interfaces
May 1996

Publication No. SG24-4691-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction	____		
Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a) Are you an employee of IBM or its subsidiaries: Yes____ No____

b) Do you work in the USA? Yes____ No____

c) Was this redbook published in time for your needs? Yes____ No____

d) Did this redbook meet your needs? Yes____ No____

If no, please explain:

_____

_____

What other topics would you like to see in this redbook?

_____

_____

What other redbooks would you like to see published?

_____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

Name	Address
Company or Organization	
Phone No.	

ITSO Redbook Evaluation
SG24-4691-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Department JN9B, Building 045
Internal Zip 2834
11400 BURNET ROAD
AUSTIN TX
USA 78758-3493

Fold and Tape | **Please do not staple** | Fold and Tape

SG24-4691-00

Cut or Fold
Along Line

Printed in U.S.A.