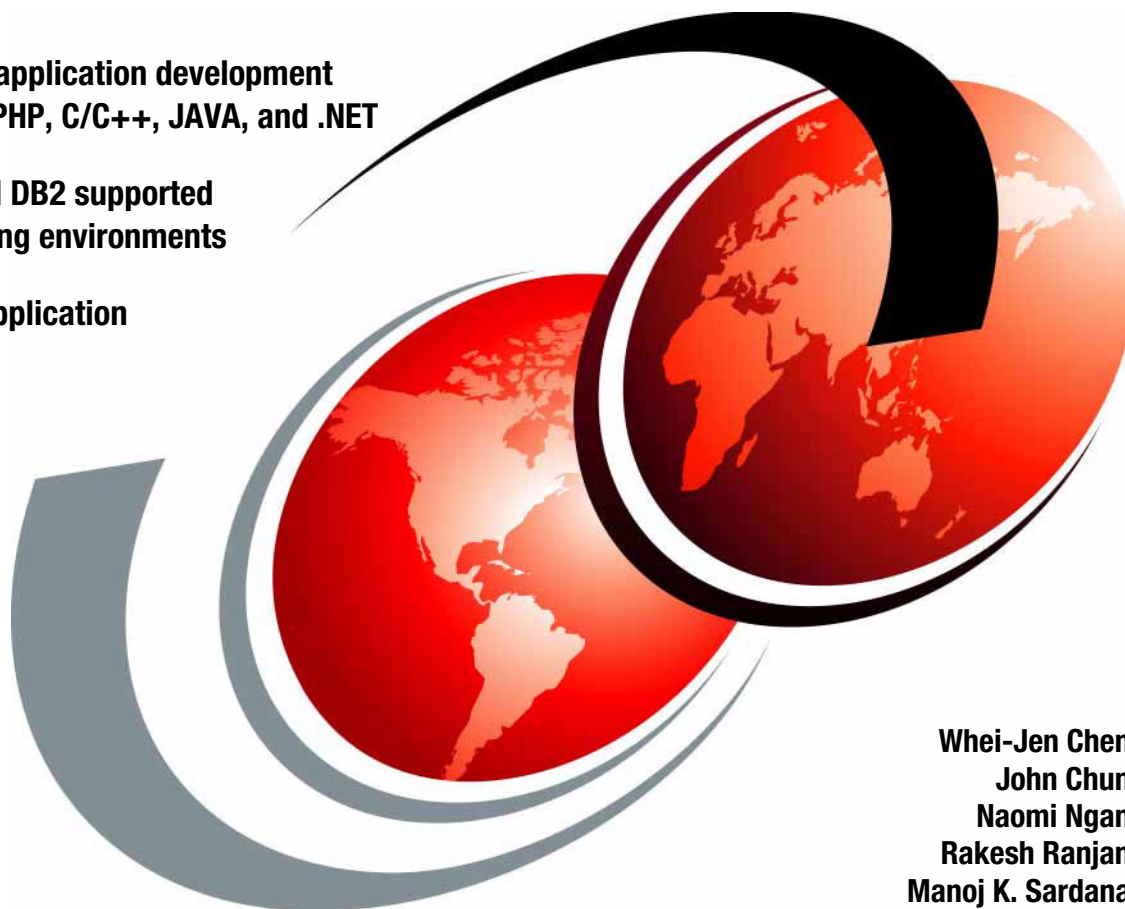


DB2 Express-C: The Developer Handbook for XML, PHP, C/C++, Java, and .NET

Learn DB2 application development
with XML, PHP, C/C++, JAVA, and .NET

Understand DB2 supported
programming environments

Practical application
examples



Whei-Jen Chen
John Chun
Naomi Ngan
Rakesh Ranjan
Manoj K. Sardana



International Technical Support Organization

**DB2 Express-C: The Developer Handbook for XML,
PHP, C/C++, Java, and .NET**

August 2006

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (August 2006)

This edition applies to DB2 Universal Database for Linux, UNIX, and Windows Version 9.

Contents

Figures	vii
Notices	ix
Trademarks	x
Preface	xi
The team that wrote this redbook	xi
Acknowledgement	xiii
Become a published author	xiv
Comments welcome	xiv
Chapter 1. DB2 application development overview	1
1.1 Application development with DB2	2
1.1.1 DB2 supported development environments	2
1.1.2 DB2 supported interfaces	3
1.2 DB2 Express-C	11
1.3 DB2 Developer Workbench	24
Chapter 2. Application development with DB2 pureXML	49
2.1 Web application: XML is the answer	50
2.2 pureXML in DB2	52
2.2.1 When to use DB2 pureXML	52
2.2.2 Designing pureXML-based applications	53
2.2.3 DB2 hybrid query engine	54
2.2.4 pureXML storage overview	55
2.2.5 SQL support for XML data (INSERT, SELECT)	58
2.2.6 Schema support	62
2.2.7 Annotated XML schema decomposition	65
2.2.8 XML query support	68
2.2.9 Constructor function (publishing functions)	77
2.2.10 XML indexing	79
2.2.11 Application support (interfaces)	83
2.2.12 Utilities and XML support	86
2.2.13 XML type support in stored procedures	90
Chapter 3. Application development with PHP	93
3.1 Application environment	95
3.1.1 Zend Framework overview	97
3.1.2 Setting up Zend Framework	99

3.2	DB2 Interface with PHP	105
3.3	Setting up Eclipse with PHP	106
3.4	Sample Web application	108
3.4.1	Integrating with databases: Zend_Db_Adapter	111
3.4.2	Zend framework: XCS	116
3.4.3	myContacts.com: An XCS application	122
3.4.4	Other Zend Framework components	139
3.4.5	Creating Web services with Zend Framework	140
3.5	Conclusion	145
Chapter 4.	Application development with C/C++	147
4.1	Overview	148
4.1.1	C/C++ development environment setup	148
4.2	Building a C/C++ application using embedded SQL	150
4.2.1	Host variables and parameter markers	151
4.3	A simple C inventory program using embedded SQL	151
4.3.1	The INVENTORY table	152
4.3.2	Precompiler source file extensions	153
4.3.3	Inventory program code template	154
4.3.4	Host variable declarations	156
4.3.5	Using db2bfd to display host variable declarations	157
4.3.6	Using db2dclgn to generate host variable declarations	157
4.3.7	Connecting to a database	158
4.3.8	Disconnecting from a database	158
4.3.9	The SQL Communications Area (SQLCA)	159
4.3.10	Quick SQLCA example	160
4.3.11	Inserting data	162
4.3.12	Retrieving data	163
4.3.13	Indicator variables	165
4.3.14	The WHENEVER Statement	166
4.3.15	Preparing SQL statements	166
4.3.16	Complete C inventory program	169
4.3.17	The SQL Descriptor Area (SQLDA)	174
4.4	Building a C/C++ application using CLI	177
4.4.1	CLI handles	177
4.4.2	The CLI driver	178
4.4.3	The CLI configuration file (db2cli.ini)	178
4.4.4	Setting up the CLI Environment	180
4.4.5	Overview of steps	182
4.5	A simple C inventory program using CLI	182
4.5.1	CLI header files	183
4.5.2	Allocating handles	183
4.5.3	Freeing handles	184

4.5.4	Connecting and disconnect to and from a database	185
4.5.5	Processing SQL statements	186
4.5.6	Complete CLI Inventory Program	188
4.5.7	Error handling	189
4.5.8	Quick SQLGetDiagRec() example	190
4.6	XML support	191
4.6.1	Embedded SQL	191
4.6.2	Call Level Interface (CLI)	196
Chapter 5.	Application development with Java	199
5.1	Application requirements	200
5.2	Drivers	200
5.3	Application example	201
5.4	java.sql package	208
5.4.1	Getting a connection	208
5.4.2	Manipulating data	209
5.4.3	MetaData	217
5.5	Stored procedure support	219
5.6	Handling large objects	222
5.7	Simple application program life cycle	225
5.8	Introduction to javax.sql package	227
5.8.1	DataSource	227
5.9	Exception handling	228
5.9.1	SQLExceptions	228
5.9.2	SQLWarning	231
5.9.3	DataTruncation	231
5.9.4	BatchUpdateException	232
5.10	Transactions	233
5.10.1	Auto commit mode	234
5.10.2	Transaction isolation level	234
5.10.3	Savepoints	237
5.11	SQL/XML and XQuery support	238
5.12	SQLj support	239
5.12.1	Getting connection context	240
5.12.2	Manipulating data	240
5.12.3	Iterators	241
5.12.4	Batch updates with SQLj	243
5.12.5	Savepoints	244
5.12.6	XQuery and SQL/XML support	244
5.12.7	Exception handling	245
5.12.8	JDBC and SQLj	245
5.13	Running the application	247
5.13.1	Running an application stand-alone	247

5.13.2 Running the application as a Web service	248
Chapter 6. Application development with .NET	251
6.1 .NET technology and ADO.NET	252
6.2 Requirements for .NET application development with DB2	253
6.3 Add-in features for Visual Studio .NET	253
6.3.1 Visual Studio 2005 Add-In: Sever Explorer integration	256
6.3.2 Visual Studio 2005 Add-In: IBM Designer	259
6.4 Data Providers for ADO.NET	267
6.4.1 Managed provider and unmanaged provider	267
6.5 Application example using ADO.NET	279
Appendix A. Setup procedure and sample data	295
A.1 Example data	296
A.2 Setting up the database	301
A.3 Setting up Apache HTTP server, PHP, and DB2 on Windows	308
Appendix B. Ruby on Rails	313
B.1 Introduction to Ruby	314
B.1.1 Getting started with Ruby programming language	315
B.2 Introduction to Ruby on Rails	315
B.2.1 DB2 9 on Rails	316
B.2.2 Further reading	316
Appendix C. Additional material	319
Locating the Web material	319
Using the Web material	319
System requirements for downloading the Web material	320
How to use the Web material	320
Related publications	321
IBM Redbooks	321
Other publications	321
Online resources	323
How to get IBM Redbooks	324
Help from IBM	324
Index	325

Figures

0-1	Left to right: Naomi, Whei-Jen, Rakesh, Manoj, and John.	xiii
1-1	Embedded SQL creation overview	5
1-2	Setup wizard: specifying response file option	14
1-3	Setup wizard: selecting DB2 features to install	15
1-4	Setup wizard: setting DB2 copy name	16
1-5	Setup wizard: specifying the location of DB2 Information Center	17
1-6	Setup wizard: setting DAS user information	18
1-7	Setup wizard: configuring DB2 instances	19
1-8	Setup wizard: DB2 instance configuration window	20
1-9	Setup wizard: preparing the DB2 tools catalog	21
1-10	Setup wizard: enabling operating system security for DB2 objects	22
1-11	Selecting new data development project	25
1-12	New Project drop-down menu	26
1-13	Data development project for a stored procedure	27
1-14	Specifying language type for stored procedure	28
1-15	Creating a Java project	29
1-16	Specifying package and SQLJ name.	30
1-17	Switching to Data perspective	31
1-18	Selecting user defined function wizard	32
1-19	XQuery Visual Builder: connecting to database.	34
1-20	DWB: creating new project	35
1-21	XQuery Visual Builder: specifying the connection	35
1-22	XQuery Visual Builder: specifying the document location	36
1-23	XQuery Visual Builder: specifying the document location	37
1-24	XQuery Visual Builder: specifying XML file name	38
1-25	XQuery Visual Builder: adding representative XML documents	38
1-26	XQuery Visual Builder: associating documents with XML columns	39
1-27	XQuery Visual Builder: XQM tab	40
1-28	XQuery Visual Builder: adding element to XQuery	41
1-29	XQuery Visual Builder: For Logic (FLWOR) window	42
1-30	XQuery Visual Builder: matches window	43
1-31	XQuery Visual Builder: For Logic (FLWOR) window with operand 1	44
1-32	XQuery Visual Builder: source tab	44
1-33	XQuery Visual Builder: Data Output tab	46
1-34	XQuery Visual Builder: XQuery results	47
2-1	Web application XML: connecting each other	51
2-2	Integrating XML and relational data	55
2-3	Creating table with XML data type	56

3-1	Eclipse	108
3-2	Movie of the Week initial page	116
3-3	XCS architecture overview	117
3-4	Navigation diagram for MyContacts.com application	124
3-5	MyContacts.com Index page	127
3-6	My Profile page showing logged-in user and his contacts	128
3-7	Create a new member profile	132
3-8	Define relationship and make contact	134
3-9	Search for Flickr image	144
4-1	CLI handles	178
4-2	Processing SQL statements	186
5-1	Different path for an SQL statement in a JDBC program	226
6-1	ADO.NET architecture	252
6-2	Adding connection in Data Explorer	256
6-3	Selecting IBM DB2 in Data Explorer	257
6-4	Providing connection information in Data Explorer	258
6-5	Starting IBM Table Designer	259
6-6	Table Designer window	260
6-7	Starting IBM View Designer	261
6-8	View Designer window	262
6-9	Starting IBM Procedure Designer	263
6-10	Procedure Designer window	264
6-11	Adding debug breakpoints in Procedure Designer	266
6-12	DB2 Data Provider	268
6-13	Sample info XML data from the CUSTOMER table	283
6-14	XML document tree for the customer info data	289

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

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

@server®	zSeries®	DB2®
Redbooks (logo)  ™	AIX®	IBM®
developerWorks®	AS/400®	OS/390®
iSeries™	Cloudscape™	Redbooks™
pureXML™	DB2 Connect™	RETAIN®
z/OS®	DB2 Universal Database™	WebSphere®

The following terms are trademarks of other companies:

Java, JDBC, JDK, J2EE, Solaris, Sun, Sun Microsystems, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Active Directory, ActiveX, Microsoft, Visual C++, Visual Studio, Windows Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

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

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

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

Preface

This IBM® Redbook will help you get started in application development with XML, PHP, C/C++, Java™, and .NET using the free IBM database management system offering DB2® Express-C V9. This book is organized as follows:

- ▶ **Chapter 1** introduces application development options for DB2 Universal Database™ for Linux®, UNIX®, and Windows®. Installation and features of DB2 Express-C and DB2 Developer Workbench are also presented.
- ▶ **Chapter 2** introduces the new pureXML™ technology in DB2 Express-C V9 and provides an overview to design and build a Web application that utilizes this technology.
- ▶ **Chapter 3** provides an easy to use framework for developing a Web application using PHP and DB2 Express-C. It discusses application environment setup and provides practical PHP application examples.
- ▶ **Chapter 4** discusses how to develop DB2 applications with C/C++ including the environment setup, the fundamentals of using embedded SQL, and XML.
- ▶ **Chapter 5** discusses how to develop Java application with DB2 Express-C including application software requirements, an in-depth description of the `java.sql` package, an overview of the `javax.sql` package, exception handling, XML/XQuery, and SQLj support.
- ▶ **Chapter 6** introduces DB2 application development using .NET. It discusses requirements for .NET application development with DB2, add-in features for Visual Studio® .NET, DB2 Data Providers available for use with .NET, and application examples using .NET and DB2 Express-C.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Whei-Jen Chen is a Project Leader at the International Technical Support Organization, San Jose Center. She has extensive experience in application development, database design and modeling, and DB2 system administration. Whei-Jen is an IBM Certified Solutions Expert in Database Administration and Application Development as well as an IBM Certified IT Specialist.

John Chun joined IBM in 2000 and has held various roles within the company. He is an IBM Certified Solutions Expert in DB2 and WebSphere. Currently, he is working as an Application Development Specialist within the IBM DB2 Advanced Support Services team where he helps customers and vendors get the best out of their applications and DB2. John also enjoys writing articles and books, which help share his experience and knowledge with all those interested in the IT Industry.

Naomi Ngan joined IBM in 2000, and worked for over three years as an application development specialist for the IBM DB2 Advanced Support team. Currently, she is a database developer, responsible for analyzing, designing, and developing Bioinformatics software for the Gallo research center at UCSF. She has in-depth knowledge of application development and tooling for DB2 and holds numerous IBM and Sun™ developer certifications in DB2, XML, WebSphere®, and Java/J2EE™.

Rakesh Ranjan is an Advisory software engineer and currently works on the DB2 XML technology development at the IBM Silicon Valley Lab in San Jose, California. He has 12+ years of software development experience ranging from midrange servers to Java and Web technology. He is also an open source enthusiast and uses XML, PHP, and frameworks for application development. You can reach Rakesh at ranjanr@us.ibm.com.

Manoj K. Sardana is a software engineer in IBM India Software Labs and works mainly on designing and implementing technology and application samples in different programming languages for showcasing the main features of DB2. His areas of expertise include XML/XQuery and JDBC™ application development. Currently, he is involved in writing application samples for SQL/XML-related and XQuery-related features for DB2 9. He has also worked closely with the development team to perform Functional Verification Testing (FVT) for Utility support for XML and Deadlock Monitor-related features.

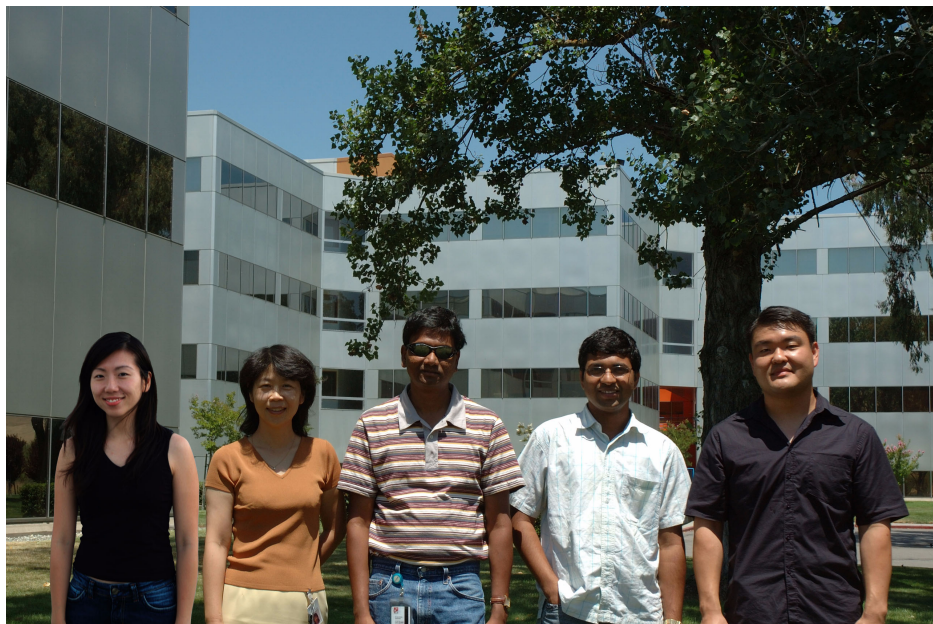


Figure 0-1 Left to right: Naomi, Whei-Jen, Rakesh, Manoj, and John

Acknowledgement

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

Grant Hutchison
Rav Ahuja
Anita Chung
Prashant Juttukonda
Michael Hvizdos
Chris Gruber
Anson Kokkat
Nancy Taiyab
IBM Toronto Laboratory

Matthias Nicola
Salvador Ledzema
Cindy Saracco
IBM Silicon Valley Laboratory

Emma Jacobs
International Technical Support Organization, San Jose Center

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an e-mail to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400



DB2 application development overview

In this chapter, we introduce application development options for DB2 Universal Database for Linux, UNIX, and Windows. We present Installation and features of DB2 Express-C and DB2 Developer Workbench.

In this chapter, we discuss the following:

- ▶ DB2 supported development environments
- ▶ DB2 supported interfaces
- ▶ DB2 Express-C
- ▶ DB2 Developer Workbench

1.1 Application development with DB2

Application programmers face constant challenges to efficiently utilize database resources to their full potential. This is especially the case with the growth of e-business and the increased use of the Internet. This redbook provides various application development options for DB2 using free distribution of DB2 Express-C.

This redbook is meant for you to use as a hands-on resource for developing DB2 UDB applications using XML, PHP, C/C++, Java, and .NET. We also cover the new Developer Workbench tool, which provides a comprehensive development environment for building data driven applications with DB2 9.

DB2 provides various flexible programming options. The DB2 product family supports all major standardized programming interfaces for data access.

The IBM DB2 9 release features native support for storing, managing, and querying XML data, which is explored throughout this book.

You should keep in mind that there are key advantages to each interface, which need to be taken into consideration when choosing the programming interface and language.

Various client side database application solutions using examples in PHP, C/C++, Java, and C# with DB2 9 Express-C are presented in later chapters.

1.1.1 DB2 supported development environments

The DB2 application development environment is composed of several elements:

- ▶ Operating system
 - AIX®
 - HP-UX
 - Linux
 - Solaris™
 - Windows

There are various conditions you should be aware of when developing application solutions on these environments. These include OS level and for Linux, kernel level as well as glibc version.

This information can be obtained from the following Web site:

<http://www.ibm.com/software/data/db2/udb/support/index.html>

- ▶ DB2 installation
DB2 V8 Application Development client or DB2 9 Client installation is required for DB2 application development.
- ▶ Database application programming interface
The interfaces include CLI/ODBC, embedded SQL, JDBC/SQLJ, OLE DB, Perl DBI, PHP, and ADO.NET. We discuss these in further detail in the next section.
- ▶ Programming language
Supported programming languages for DB2 include C, C++, COBOL, FORTRAN, Java, Perl, PHP, REXX, and .NET.
- ▶ Transaction manager and development tools
Keep in mind that using the XA (distributed transaction processing) interface requires transaction manager.

There are a number of development tools available for DB2 database application development. We discuss IBM DB2 Development Add-In for Visual Studio (Visual Studio 2005 Add-In) and the Developer Workbench in this book.

1.1.2 DB2 supported interfaces

IBM DB2 provides various programming interfaces for data access. These interfaces provide functions and methods that you can use to perform various interactions with the database.

Administrative API

The DB2 Administrative API is primarily used for issuing DB2 administrative commands. It allows you to develop applications that you can use to administer and monitor DB2 instances and databases.

All the administrative API functions are derived from DB2 Command Line Processor (CLP) commands. You can find more information regarding DB2 API functions in the *Administrative API reference*, SC10-4231.

Embedded SQL

The SQL statement can be embedded within a host language where SQL statements provide the database interface while the host programming language provides all remaining functionality.

The nature of embedded SQL applications requires a vendor specific precompiler first preprocesses the application code, then the resulting host language code is compiled and linked directly with the vendor's library.

DB2 supports C/C++, FORTRAN, COBOL, and Java (SQLJ) programming languages for embedded SQL.

SQL statements in embedded applications are independent of the host programming language used.

Precompilers or SQLJ translators are required for embedded code (to generate the necessary packages or the serialized file for SQLJ) prior to generating the binaries through the native compiler. Precompilers for C, C++, COBOL, and FORTRAN as well as the SQLJ translator are provided with DB2 application development components.

A precompiler processes the source file to separate SQL statements from non-SQL host languages, which are surrounded by special delimiters to generate native host language code and a package.

Embedded SQL applications can be categorized into two separate categories:

- ▶ Static embedded SQL

In embedded SQL, you have to specify complete SQL statement structure. This means that all database objects (including columns and table) must be fully known at the precompile time with the exception of objects referenced in the SQL WHERE clause. However, all host variable data types must be still known at the precompiler time.

- ▶ Dynamic embedded SQL

When not all of the database objects in the SQL statement are known at precompile time, use dynamic embedded SQL. The dynamic embedded SQL statement accepts a character string host variable and a statement name as argument. These character string host variables serve as placeholders for future SQL statements to be executed.

There are some differences in syntax between static and dynamic SQL statements. In dynamic SQL statements, the following conditions exist:

- ▶ Comments are not allowed.
- ▶ The statement is not prefixed by EXEC SQL.
- ▶ The statement cannot end with the statement terminator except in the CREATE TRIGGER statement, which can contain a semicolon.

The dynamic embedded SQL is the ideal option for creating embedded SQL applications when the user does not have complete information about all

underlying database objects at precompile time, wants to always use the most optimized access path based on current database statistics, or authorization of the SQL statement needs to be determined at runtime.

Figure 1-1 demonstrates a general overview of embedded SQL creation.

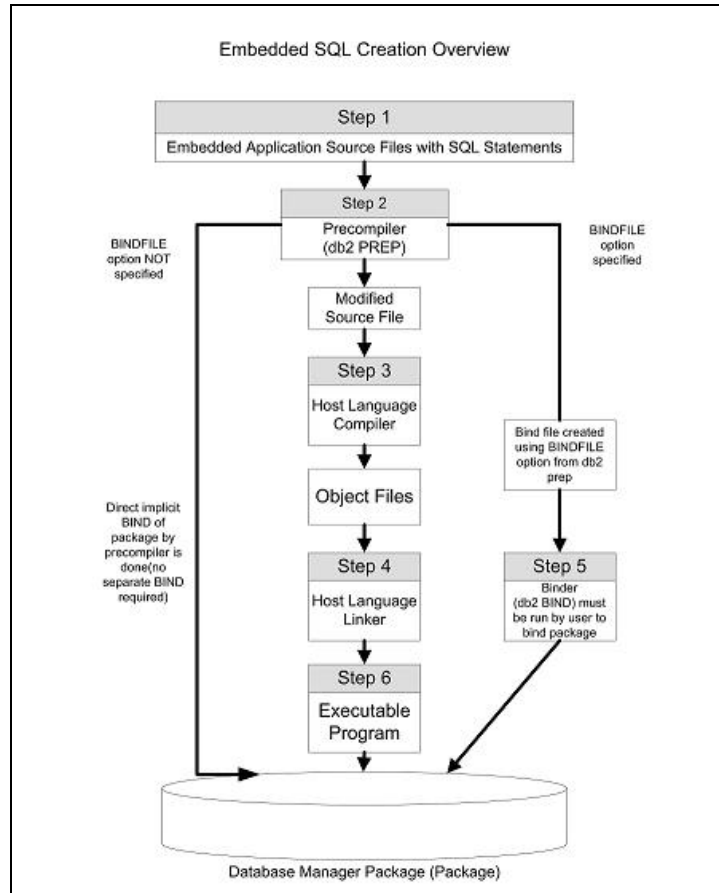


Figure 1-1 Embedded SQL creation overview

Driver support

More common application solutions are developed using drivers. When accessing a database using various available drivers, driver manager is usually involved. The driver manager provides a set of industry standard interfaces (APIs) to access a data source using data source specific drivers. Applications utilizing drivers are compiled and linked with the driver manager's libraries to invoke standardized APIs.

DB2 currently provides support for a large number of drivers, including CLI/ODBC, ADO and OLEDB, JDBC, SQLJ, PERL DBI, and .NET data provider.

CLI/ODBC

As part of the X/Open standard, Call Level Interface (CLI) and Open DataBase Connectivity (ODBC) standards had the same origin. The ODBC standards provide a set of interfaces for accessing the database.

DB2 CLI driver can be used on its own to access a DB2 database or as an ODBC driver. The DB2 CLI driver is an ODBC 3 compliant driver and contains further functionality that is not specified in ODBC standards. In order to utilize additional functionality in CLI driver, the application program needs to be linked directly to CLI driver without the use of the ODBC driver manager.

Perl Database Interface (DBI)

Perl is part of Open Source Standard and is one of the popular choices for use with Web services through the Common Gateway Interface (CGI). IBM DB2 provides support for Perl Scripts using Database Interface (DBI). DBI provides a set of standard class methods to access data sources using drivers called *Database Drivers (DBD)*. In order to develop Perl application solutions, you need to obtain Perl, the DBI module, and the DBD:DB2 driver from the Comprehensive Perl Archive network:

<http://www.cpan.org>

Building and installing the DBD:DB2 module requires the following:

► For Linux and UNIX:

To build and install the DBD:DB2 module, you must have:

- Perl 5.005_03 or later.
- DB2 V8 Application Development Client or DB2 9 client.
- A supported C compiler as documented under “Supported operating systems” on the Application development Web page.
- Set the DB2_HOME environment variable to the location of your DB2 instance. For example:

```
bash# export DB2_HOME=/home/db2inst1/sql1ib
```

- Install the DBI module:

```
bash# perl -MCPAN -e 'install DBI'
```

- Install the DBD:DB2 module:

```
bash# perl -MCPAN -e 'install DBD:DB2'
```

► For Windows:

If you are using the ActiveState Perl distribution on Windows, you can install a binary version of the DBI and DBD::DB2 modules.

Prerequisites:

- ActivePerl 5.8 or later
- DB2 client, Version 8.1 or later

For example, if you have ActivePerl 5.8.7 installed, you can install the DBD::DB2 modules as follows:

1. Install the DBI module by issuing the following command (the DBI module is a prerequisite for the DBD::DB2 module):

```
ppm install  
http://www.cpan.org/authors/id/H/HO/HOYMICH/db2/perl58/DBI-1.45.ppd
```

2. Install the DBD::DB2 module by issuing the following command:

```
ppm install  
http://www.cpan.org/authors/id/H/HO/HOYMICH/db2/perl58/DBD-DB2-0.78.ppd
```

To uninstall the DBD-DB2 module, issue:

```
ppm uninstall DBD-DB2
```

Alternatively, you can download the corresponding DBD-DB2.ppd and .tar.gz files from:

<http://www.cpan.org/authors/id/H/HO/HOYMICH/db2/>

Install them locally by issuing the following command:

```
ppm install DBD-DB2.ppd
```

When the DBD::DB2 module is successfully installed, you can access documentation and a sample application by issuing the following command:

```
perldoc DBD::DB2
```

OLE DB

OLE DB is a data access service that was introduced with Microsoft®'s ActiveX® Data Objects (ADO). ActiveX Data Objects are a set of classes by which applications can access data from multiple sources that utilize a given layer. It provides consistent access to data sources exposed through OLE DB, XML, or third-party .NET data provider.

IBM's OLE DB driver is called IBMDADB2. If IBMDADB2 is not explicitly specified, Microsoft's OLE DB driver (MSDASQL) will be utilized by default.

MSDASQL allows clients utilizing OLE DB to access third party (non-Microsoft SQL server) data sources using ODBC driver but does not guarantee full functionality of OLE DB driver.

JDBC driver

Java is one of the most popular choices for application solutions. Write once, run everywhere is the theme of JAVA, which reduces development time and has made this technology an ideal choice for many.

The data source access is achieved through the JDBC programming interface in Java. JDBC driver specification defines four types of driver architectures:

▶ **Type 1**

Drivers that implement the JDBC API as a mapping to another data access API, such as Open Database Connectivity (ODBC). Drivers of this type are generally dependent on a native library, which limits their portability. DB2 does not provide type 1 driver.

▶ **Type 2**

Drivers that are written partly in the Java programming language and partly in native code. The drivers use a native client library specific to the data source to which they connect. Because of the native code, their portability is limited.

▶ **Type 3**

Drivers that use a pure Java client and communicate with a server using a database-independent protocol. The server then communicates the client's requests to the data source. DB2 no longer includes a type 3 driver.

▶ **Type 4**

Drivers that are pure Java and implement the network protocol for a specific data source. The client connects directly to the data source.

DB2 V8 provides type 2 (APP driver and JCC type 2), type 3 (NET driver), and type 4 (JCC type 4) drivers with APP and NET drivers being deprecated from V8 GA but still shipped (for V8).

DB2 9 provides type 2 (APP driver and JCC type 2) and type 4 (JCC type 4) drivers. The JDBC type 3 driver (NET) has been discontinued and is not shipped with V9. The DB2 JDBC type 2 (APP) driver was deprecated in DB2 V8 and will remain deprecated in V9. Support for DB2 JDBC type 2 (APP) will be removed in a future release.

Note that there have been no functional enhancements on deprecated DB2 JDBC type 2 (APP) and DB2 JDBC type 3 (NET) drivers since DB2 V7.

All future Java application development on DB2 UDB should be done using DB2 JCC type 2 or DB2 JCC type 4 drivers.

Table 1-1 summarizes the DB2 JDBC driver support on V8 and V9.

Table 1-1 DB2 JDBC driver

	V8		V9	
	Shipped	Deprecated	Shipped	Deprecated
JDBC type 2 (APP)	Yes	Yes	Yes	Yes
JDBC type 3 (NET)	Yes	Yes	No	Yes
JCC type 2	Yes	No	Yes	No
JCC type 4	Yes	No	Yes	No

Note: JCC in V9 is JDBC 3 compliant.

SQLJ

Along with host language embedded SQL type applications, there are also embedded Java applications, better known as SQLJ programs. SQLJ is a method for accessing DB2 from a Java application that supports static execution. Again, the benefits of a static execution are reduced resource consumption, improved diagnostics, improved security, and greater repeatability of SQL performance due to static access paths and plans. Everything you need to get from the data is already in the package bound at bind time.

SQLJ provides performance benefits of static query execution by embedding SQL queries into Java applications. SQLJ still utilizes the JDBC driver to access data source and is the layer above JDBC. SQLJ translator is used to process SQLJ source files with the extension .sqlj. It translates .sqlj source files into .java files and an SQLJ serialized profile into a form of .ser file. The serialized file contains all the SQL statements in original SQLJ source file. The translated resulting .java file will contain calls to SQLJ run-time libraries in place of SQL statements. In order to bind the application statically to a DB2 database, you use the DB2 profile customizer tool called db2sqljcustomize. The db2sqljcustomize connects and binds a package on the target database using the serialized profile. The package bound in the target database using db2sqljcustomize will contain sections which correspond to each SQL query in the serialized profile.

Commands associated with SQLJ:

▶ **sqlj:**

sqlj is the translator that takes an embedded SQLJ program and creates a .ser file used for binding and a .java file that will also be compiled into byte code, as typical Java programs are compiled.

▶ **db2sqljcustomize:**

This command will take the .ser file from the **sqlj** step, connect to the database against which the application will be run, and bind four bind files for this application, all with different isolation levels.

▶ **db2sqljbind:**

This command can be used to rebind this application against other databases; for example, it can be used for moving the application from the test to the production database.

The following packages need to be imported for SQLJ:

```
import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;
```

PHP

PHP: Hypertext Preprocessor (PHP) is an interpretive programming language intended for Web application development. IBM supports access to DB2 Database from PHP applications through two extensions:

▶ **ibm_db2:**

The **ibm_db2** extension offers a procedural application programming interface to create, read, update, and write database operations in addition to extensive access to the database metadata. It can be compiled with either PHP 4 or PHP 5.

▶ **PDO_ODBC:**

The **PDO_ODBC** is a driver for the PHP Data Objects (PDO) extension that offers access to DB2 database through the standard object-oriented database interface introduced in PHP 5.1. It can be compiled directly against DB2 libraries.

.NET data provider

The .NET developers have choices of incorporating ODBC .NET Data provider, OLE DB .NET Data provider, or DB2 .NET Data provider (native provider). We recommend that you first consider DB2 .NET Data provider when it comes to .NET Application development. There are a number of enhancements made in DB2 .NET Data provider in V9 for native XML support:

- ▶ The ODBC .NET Data provider makes ODBC calls to DB2 data source using DB2 CLI driver. It has same keyword support and restrictions as that of DB2 CLI driver and can be used only with .NET Framework Version 1.1 or Version 2.0. This utilizes IBM DB2 ODBC (thus CLI) driver.
- ▶ The OLE DB .NET Data provider uses IBM DB2 OLE DB Driver (IBMDADB2). It has same keyword support and restrictions as that of DB2 OLE DB driver and can be used only with .NET Framework Version 1.1 or Version 2.0. This utilizes IBM DB2 OLE DB (IBMDADB2) driver.
- ▶ The DB2 .NET Data provider extends DB2 support for the ADO.NET interface. The DB2 managed provider implements the same set of standard ADO.NET classes and methods and it is defined under IBM.DATA.DB2 namespace. We will use DB2 .NET Data provider for the sample application example in Chapter 6, “Application development with .NET” on page 251.

1.2 DB2 Express-C

DB2 Express-C is a version of DB2 Universal Database Express Edition (DB2 Express) for the community, which is completely free to download, develop, deploy, test, run, embed and redistribute. It provides the same core data server features and development interfaces as well as system limits as DB2 Express in a smaller package. DB2 Express-C is available for Linux and Windows running 32-bit or 64-bit hardware with up to 2 CPUs and 4 GBs of memory.

Support for DB2 Express-C is made available through forum:

http://www.ibm.com/developerworks/forums/dw_forum.jsp?forum=805&cat=19

DB2 Express-C can be seamlessly upgraded to DB2 Express, Workgroup, and Enterprise Server Edition without database or application modification.

DB2 Express-C is free to download from:

<http://www.ibm.com/software/data/db2/udb/db2express/>

DB2 Express-C installation

Before installing DB2 database, consult the system requirements listed in following Web site:

<http://www.ibm.com/software/data/db2/udb/sysreqs.html>

The user installing DB2 should have System Administrator authority on the system where the installation is to occur.

In a Linux system, you can use `uname -a` to determine the current kernel level and `rpm -qa | grep glibc` for the glibc version.

There are a number of common and alternate methods for installing DB2:

- ▶ **DB2 Setup Wizard**
GUI installer available on Linux (requires X window server) and Windows operating systems. It can be used to create instances and response files.
- ▶ **Response file install**
Automated install using the response file to avoid user interaction during install and to ensure the same install options are used for multiple installs.
- ▶ **db2_install script**
It is only available on Linux and installs all components for the DB2 product with English interface support. Additional language support can be selected using `-L` parameter. It allows more control over the setup process and less over the installation process. No user and group creation or configuration will occur.
- ▶ **Third-party deployment tools**
Installation method for Windows. Used for mass client deployment using Windows Active Directory®, Windows System Management Server, or Tivoli.

Table 1-2 summarizes DB2 installation methods.

Table 1-2 DB2 installation methods

Installation method	Windows	Linux or UNIX
DB2 Setup Wizard	Yes	Yes
db2_install Script	No	Yes
Response file installation	Yes	Yes
Third-party deployment tools	Yes	No

Installation steps using DB2 Setup Wizard

DB2 Setup Wizard is the most common method of installing DB2. It provides a graphical user interface to step you through the DB2 installation process:

1. Execute `setup.exe`, which displays the welcome window.
The welcome window has several choices which include: Installation Prerequisites, Release Notes, Migration Information, Install a Product, and Exit. Upon choosing **Install a Product**, two installation options are presented. One is for DB2 Express-C, which installs the database server component and another is for DB2 Client, which only consists of DB2 client component. Choose **DB2 Express-C install**, which starts the Setup Wizard. It will take few moments for the Next button to become available. Click **Next** once it becomes available.
2. Accept the terms in the license agreement to continue.
3. Several installation types are presented: Typical, Compact, and Custom. Choose **Custom** to include all application development features.
4. Now you are offered Installation options regarding response file creation. Installation can proceed with or without response file generation or you can specify the response file for future installations.

The response file path can be changed from the specified default. For this scenario, specify installation with response file. See Figure 1-2.

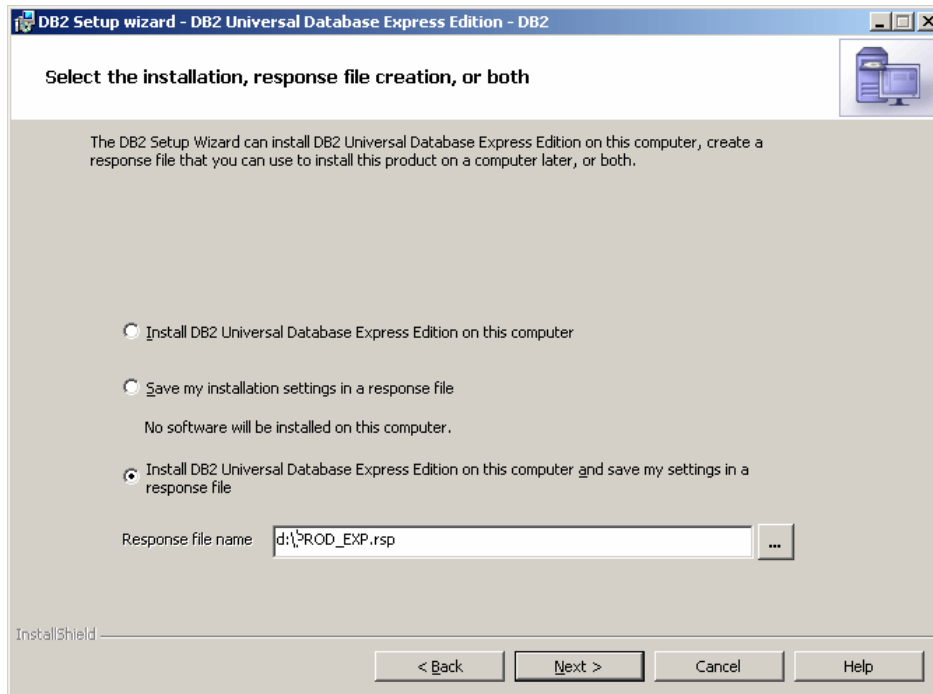


Figure 1-2 Setup wizard: specifying response file option

5. The Select features to install window is displayed (Figure 1-3). All Application development tools should be chosen by default. You can change the default installation path on this window. For this walkthrough, we choose the default.

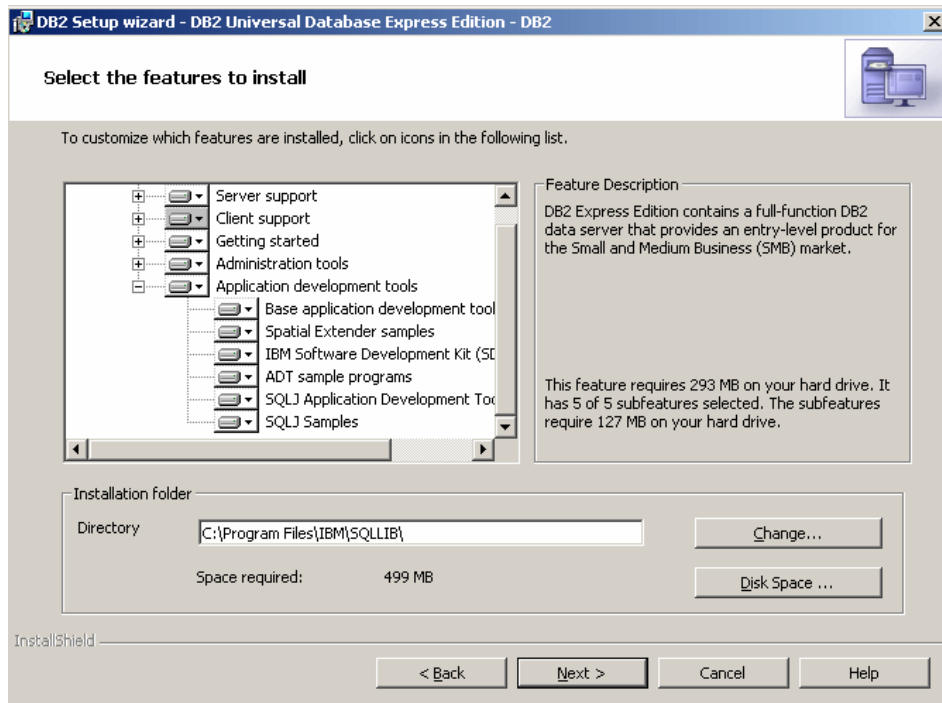


Figure 1-3 Setup wizard: selecting DB2 features to install

6. Select a language to install.

7. Set the DB2 copy name window is next (Figure 1-4). This is used to identify a location where DB2 products are installed on the computer.

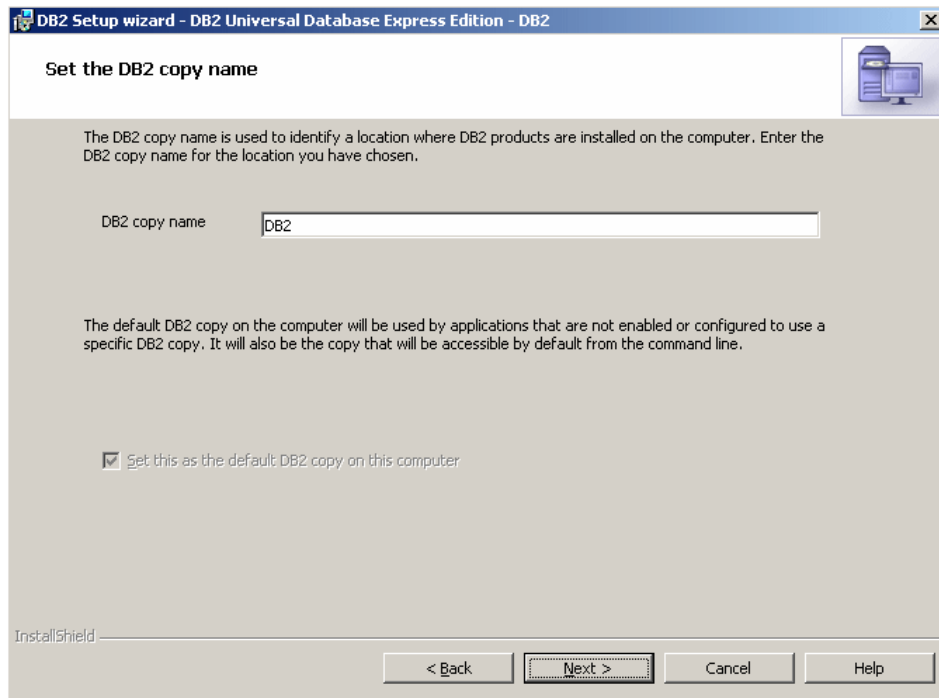


Figure 1-4 Setup wizard: setting DB2 copy name

- Specify the location of the DB2 Information Center. Choices are IBM Web site or On intranet server. We accept the default of the external IBM Web site. See Figure 1-5.

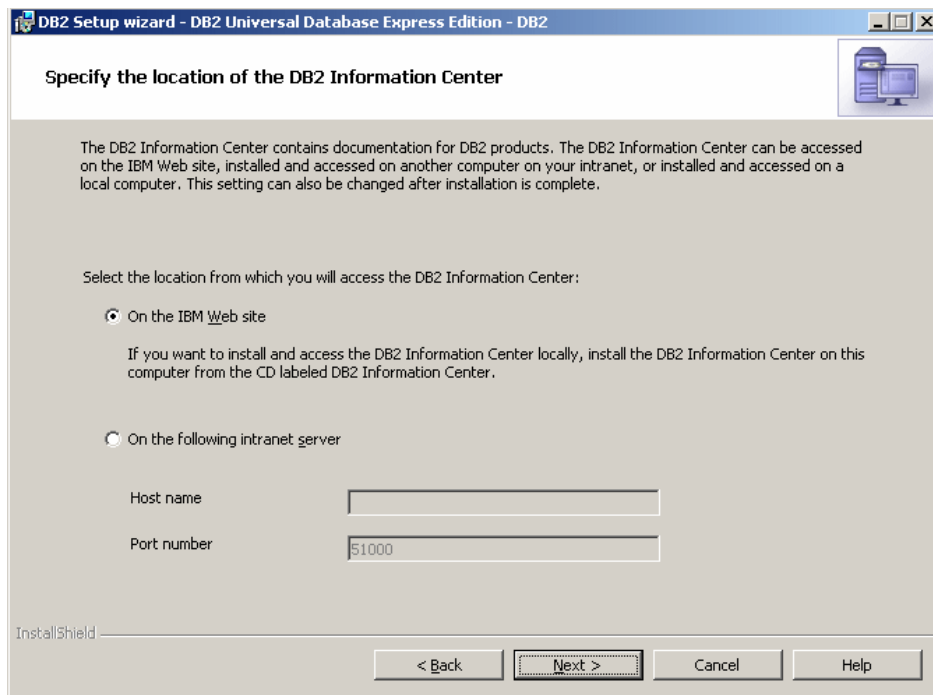


Figure 1-5 Setup wizard: specifying the location of DB2 Information Center

- Next, set the user information for DB2 Administrator server (DAS). This user ID is used to start DB2 Administration server, which responds to various DB2 administration tools and Configuration Assistant (CA).

In this section of the wizard, you can also specify the option to use the same user ID for other services. See Figure 1-6.

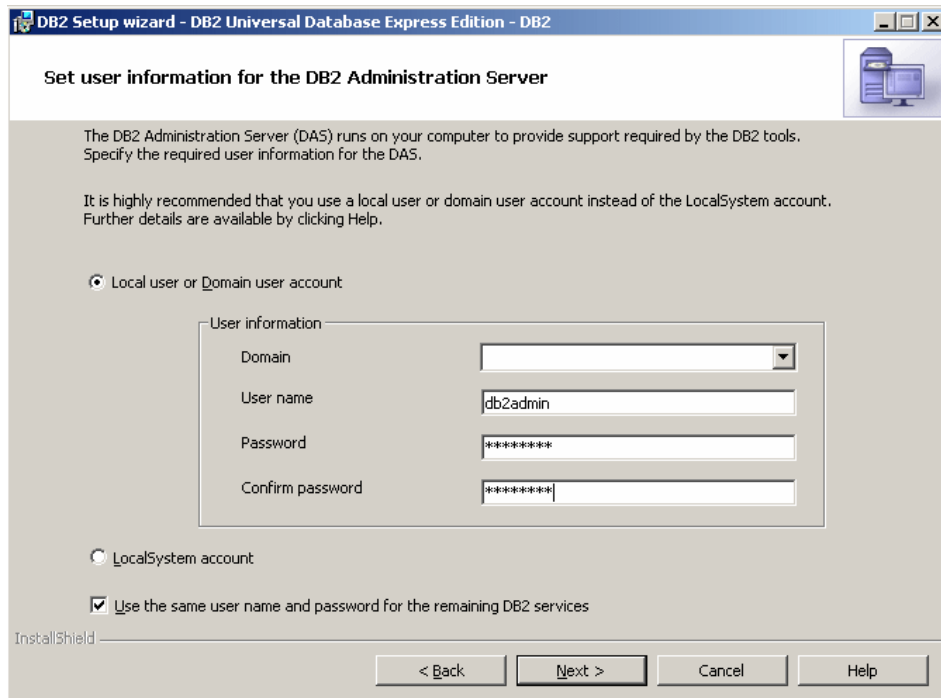


Figure 1-6 Setup wizard: setting DAS user information

10. The Configure DB2 instance window allows creation of DB2 instances. In Windows, by default, a DB2 instance called DB2 will be created. See Figure 1-7.

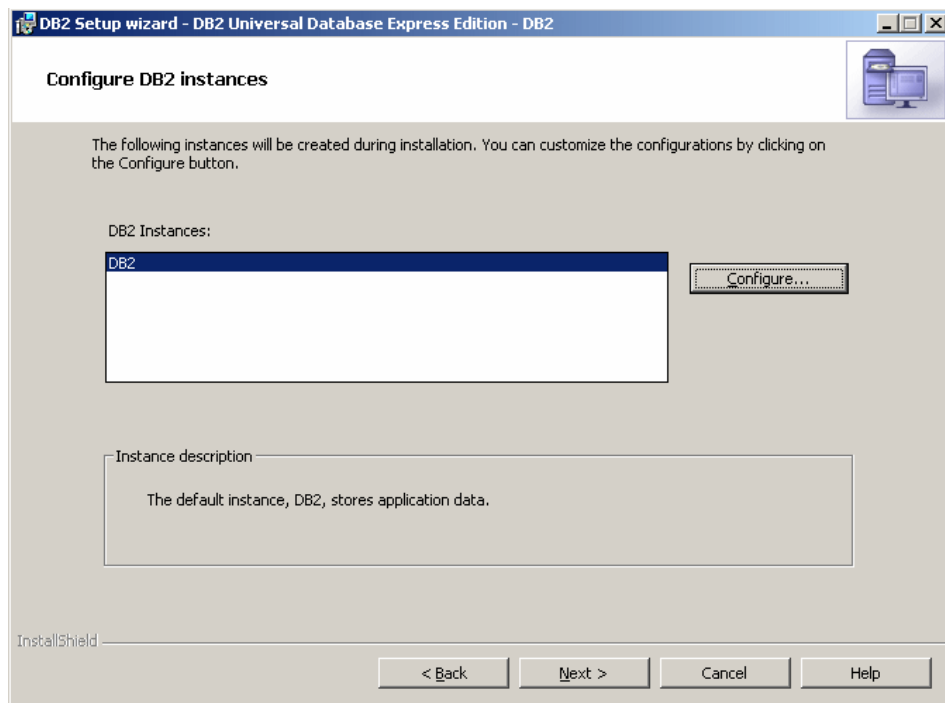


Figure 1-7 Setup wizard: configuring DB2 instances

The Configure button provides the option to set or alter the service name and listener port number associated with the given instance. See Figure 1-8.

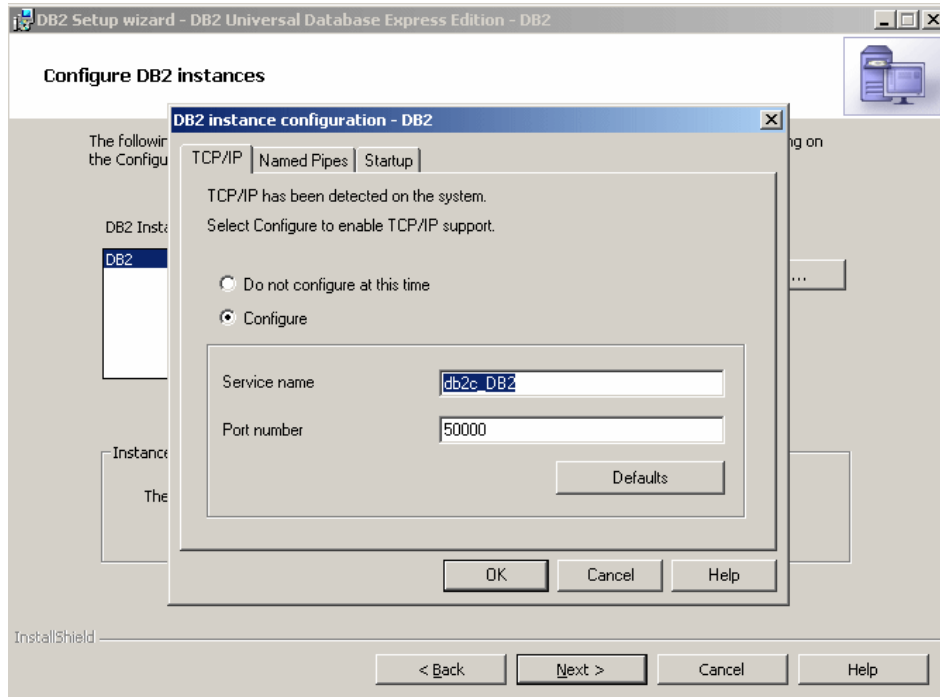


Figure 1-8 Setup wizard: DB2 instance configuration window

11. The next step involves preparing the DB2 tools catalog. The Tools catalog will be required for DB2 Task Center and Scheduler.

This step gives you the option to create the tools catalog in a new or an existing database. See Figure 1-9.

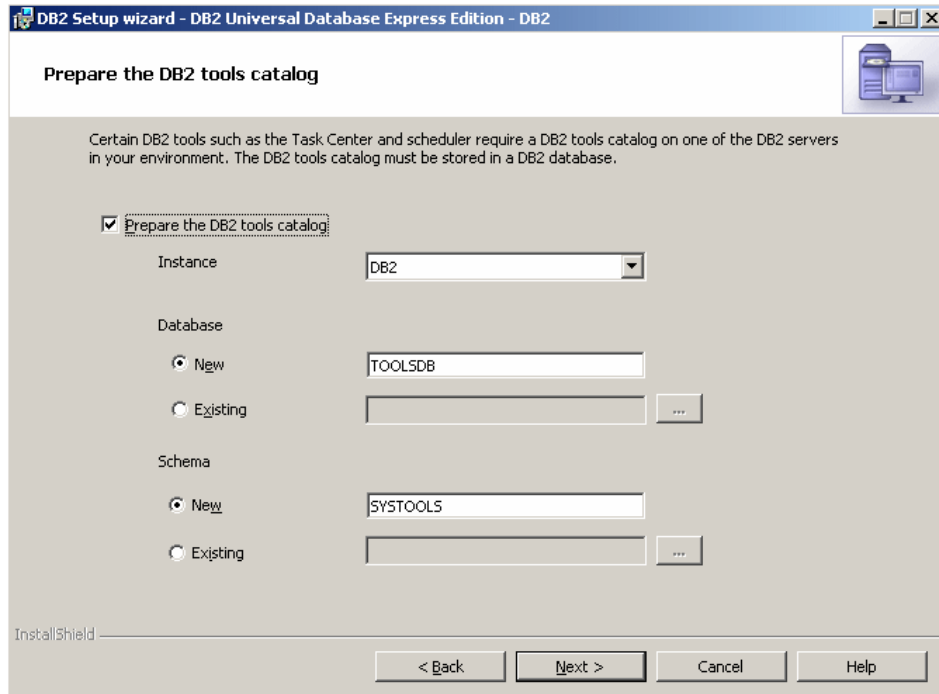


Figure 1-9 Setup wizard: preparing the DB2 tools catalog

12. The Set up notifications window provides you the option to set up DB2 send notifications automatically to the SMTP server when a database needs attention. For our walkthrough, we uncheck this option.

13. Enable operating system security for DB2 objects window (Figure 1-10) provides you the option to enable operating system security, which is chosen by default. With this security option, only users who belong to DB2 users group (DB2USERS) and DB2 administrator group (DBADMINS) can access DB2 objects.

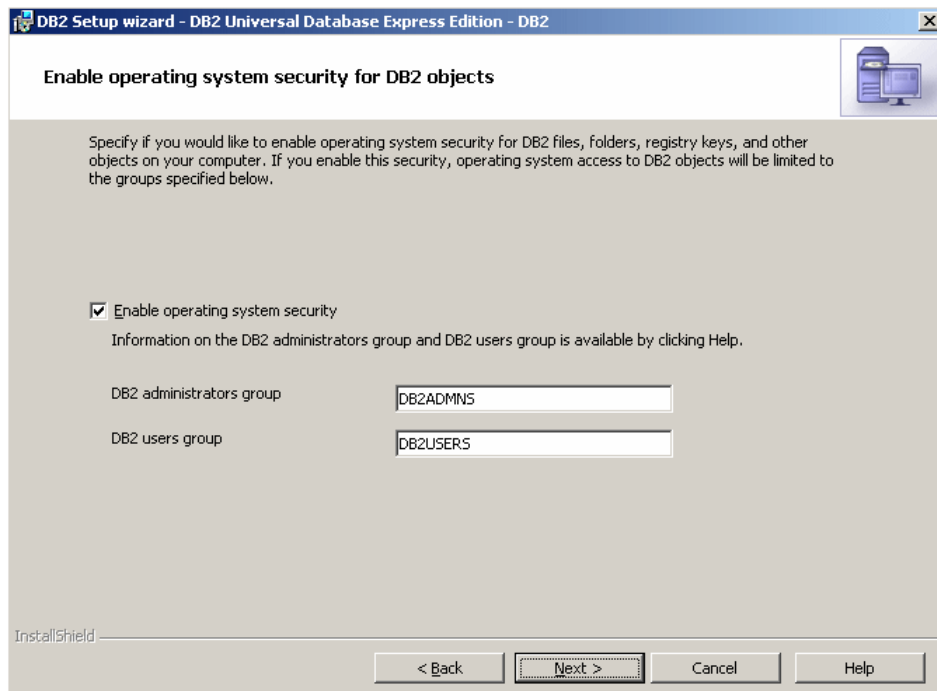


Figure 1-10 Setup wizard: enabling operating system security for DB2 objects

14. Finally, DB2 Setup wizard starts copying the files and creates the response file.
15. After the successful installation of DB2 Express-C, you can create the sample database.

Installation using response file

On the Windows platform, enter following command:

```
setup -u my.rsp
```

On the Linux platform, use the following command:

```
db2setup -r <responsefile_directory>/<response_file>
```

Installation using db2_install script

On a Linux system, log in as a user with root authority.

If DB2 Express-C has been downloaded from the previously mentioned URI, it needs to be decompressed and untarred prior to installation:

1. Decompress the product file:

```
gzip -d db2exc_91_LNX_x86.tar.gz
```

2. Untar the product file:

```
tar -xvf db2exc_91_LNX_x86.tar
```

3. Change directory into the product directory where db2_install can be found and enter the following command:

```
./db2_install
```

Sample database creation

There are several options for creating the sample database.

You can create the sample database with just the tables in V8 sample SQL database objects, XML database objects, or both.

The graphical option prompts you for the creation of the database once installation is complete and provides graphical options for creating a sample database.

The command line option involves the **db2samp1** command. Key options are:

- ▶ **-sq1**: Create SQL database objects and data
- ▶ **-xm1**: Create XML database objects and data
- ▶ **-v8**: Create the SAMPLE database from DB2 UDB V8

Considerations for DB2 Express-C

The following constraints are present in DB2 Express-C:

- ▶ Maximum processors: 2
- ▶ Maximum addressable memory: 4 GB
- ▶ Database Partitioning Feature: N/A
- ▶ Connection Concentrator: N/A
- ▶ DB2 Geodetic Extender: N/A
- ▶ Query Patroller: N/A
- ▶ Net Search Extender: N/A

- ▶ Exclude options such as HADR, DB2 Connect™ support, and Warehouse Manager tools
- ▶ DB2 Express-C community assistance can be found at the following online forum:

http://www.ibm.com/developerworks/forums/dw_forum.jsp?forum=805&cat=19

1.3 DB2 Developer Workbench

DB2 Developer Workbench (DWB) is a comprehensive development environment available for use with DB2 9. It is used to create, edit, debug, test, and deploy DB2 stored procedures and user defined functions, along with SQL, SQLJ, and XQuery APIs.

Available for Windows 32-bit and Linux 32-bit environments, it was formerly packaged with DB2 as the Development Center. This new DB2 Developer Workbench is based on Eclipse and features new XQuery Visual Builder.

You can download DB2 Developer Workbench from the following URL:

<http://www.ibm.com/software/data/db2/udb/db2express/download.html#viper>

Key features of DB2 Developer Workbench

DB2 Developer Workbench provides a universal toolset for DB2 solutions development based on the Eclipse project. In addition to the functionality of the previous Development Center, additional new features have been added. These include:

- ▶ Addition of Developer Workbench information center and tutorials.
- ▶ Ability to migrate existing Development Center projects.
- ▶ Compare routines within a data development project.
- ▶ Ability to deploy routines to different servers on various platforms.
- ▶ Develop Java and SQL stored procedures.
- ▶ Develop SQLJ applications.
- ▶ Stored procedure debugger for SQL or Java stored procedures.
- ▶ XML support:
 - XML functions
 - XML data type
 - XML schema registration
 - XQuery builder

Creating New Project in DB2 Developer Workbench

You can start a new development project in DB2 Developer Workbench using two methods.

- ▶ Select **File** → **New**, then select from a list of choices. Figure 1-11 shows the resulting window output.

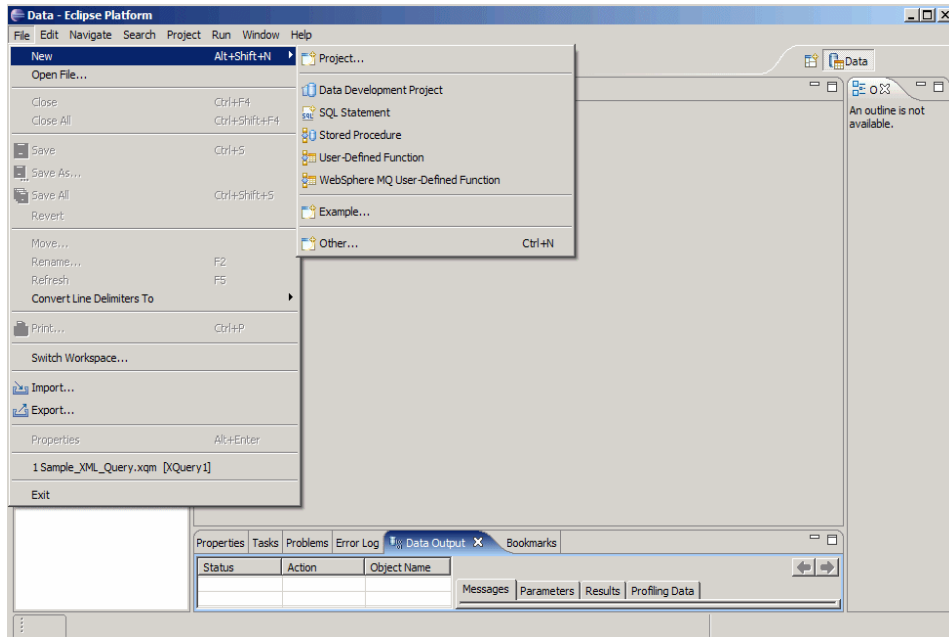


Figure 1-11 Selecting new data development project

- ▶ You can make these same choices by clicking **New Project** from the tool menu (first button on the left). Figure 1-12 on page 26 shows window output from selecting New Project.

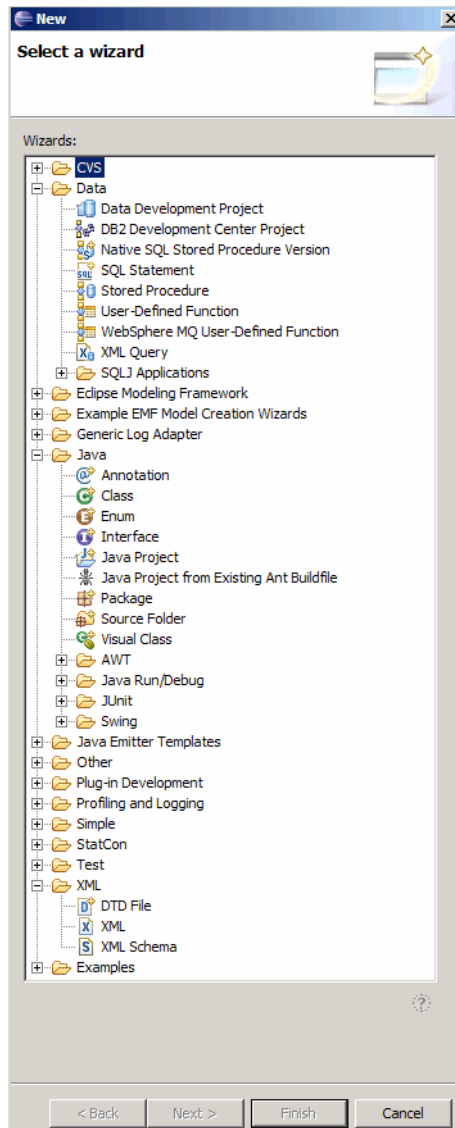


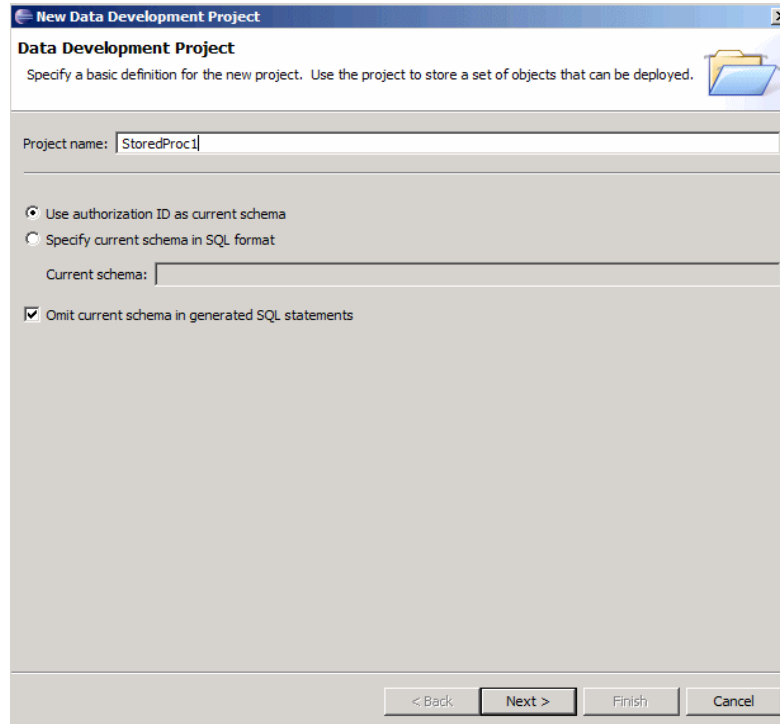
Figure 1-12 New Project drop-down menu

Creating stored procedure

To create a stored procedure:

1. Start a new stored procedure project by selecting **File** → **New** → **Stored Procedure**.

This prompts you for the project name. If new project creation is specified, you can specify the current schema as shown in Figure 1-13.



New Data Development Project

Data Development Project

Specify a basic definition for the new project. Use the project to store a set of objects that can be deployed.

Project name:

Use authorization ID as current schema
 Specify current schema in SQL format

Current schema:

Omit current schema in generated SQL statements

< Back Next > Finish Cancel

Figure 1-13 Data development project for a stored procedure

2. In the following window (Figure 1-14), choose **Create a new connection** or **Use an existing connection**. Once your choice is made, click **Finish** to return to the Specify a Project window.

Upon choosing the newly-created project name, the name of the stored procedure and the language of the procedure need to be specified as shown in Figure 1-14.

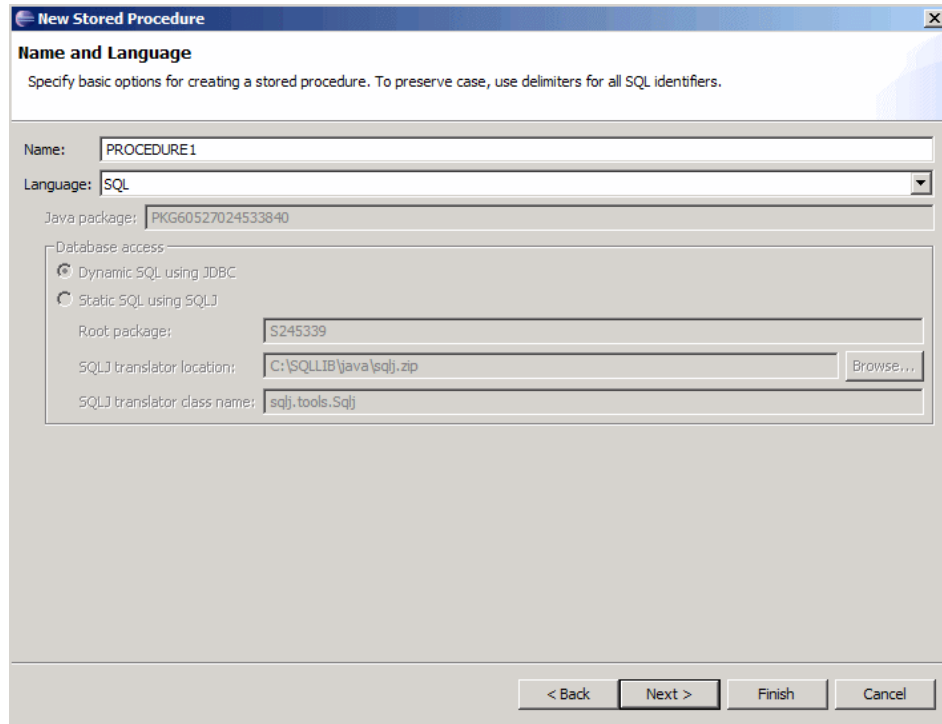


Figure 1-14 Specifying language type for stored procedure

3. Once selections are made, you are prompted for the password to connect to the target database (if the connection does not already exist). The wizard then steps you through the creation of the procedure.

Creating SQLJ

Before invoking the wizard, you need to create the Java project prior to the SQLJ creation using the wizard (see Figure 1-15):

1. Click **File** → **New** → **Other**, or click **New toolbar**. Expand the Java folder and select **Java Project**. Specify the new project name. Here you can specify other options such as JDK™ compliance level. Click **Finish** to complete the creation of the new Java Project as shown in Figure 1-15 on page 29.

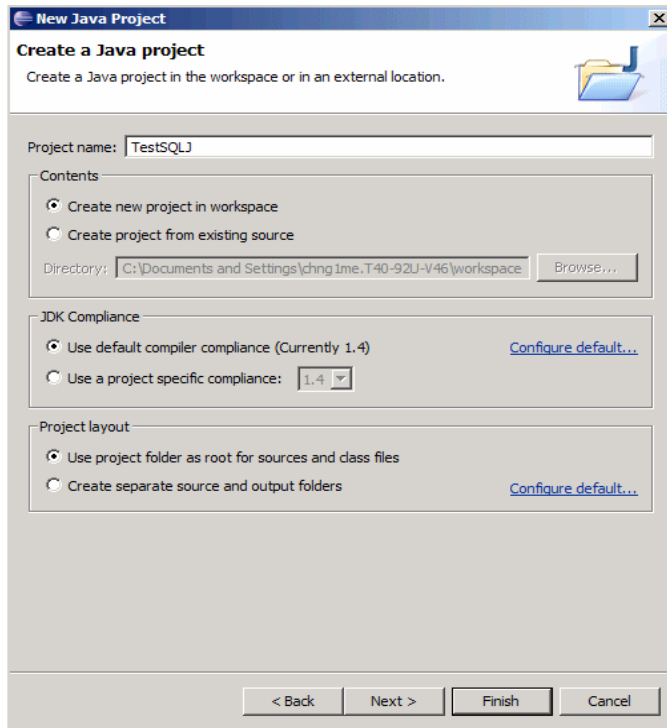


Figure 1-15 Creating a Java project

- Using the newly created or the existing Java project (highlight the project), click **File** → **New** → **Other**, or click **New toolbar**. In the New window, expand the Data folder, and then expand the SQLJ Applications folder to show the available SQLJ choices. Click **SQLJ File**, and then click **Next**.

3. You can now specify Package name and SQLJ name as shown in Figure 1-16. The SQLJ name should follow the Java type convention and should start with an uppercase character.

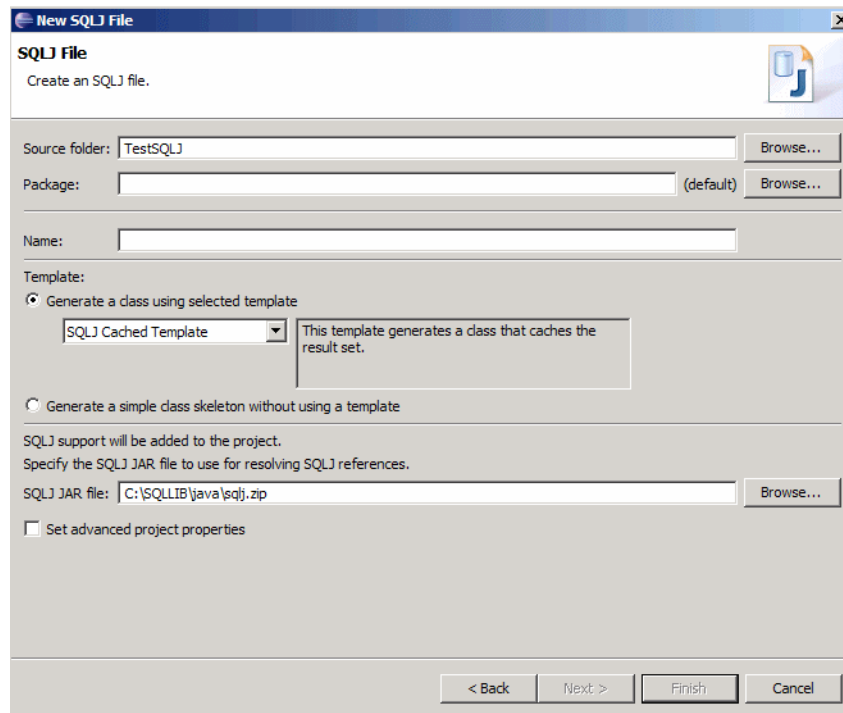


Figure 1-16 Specifying package and SQLJ name

4. Complete the wizard steps.

Creating a user defined function (UDF)

Before creating UDFs, you need to create the data development project to store the given database object:

1. Start the new Data Development project by selecting **File** → **New** → **Data development**. This will prompt you for the project name. If new project creation is specified, you can specify the current schema as shown in Figure 1-13 on page 27.
2. In following window (Figure 1-17 on page 31), you can choose to **Create a new connection** or **Use an existing connection**. Once the choice is made, clicking **Finish** returns you to the workspace window.

The data development project now displays in the Data Project Explorer view.

Now, you can use the New User Defined Function wizard to create DB2 user defined functions (UDFs) in SQL.

3. Switch to the Data perspective using **Window** → **Open Perspective** → **Other**, then select **Data**. Output of the Select Perspective window is shown in Figure 1-17.

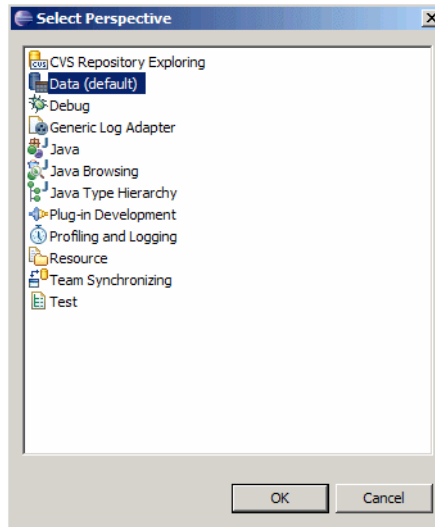


Figure 1-17 Switching to Data perspective

4. In the Data Project Explorer, right-click the **User Defined Functions** folder in a data development project, and click **New** → **User Defined Function** as shown in Figure 1-18.

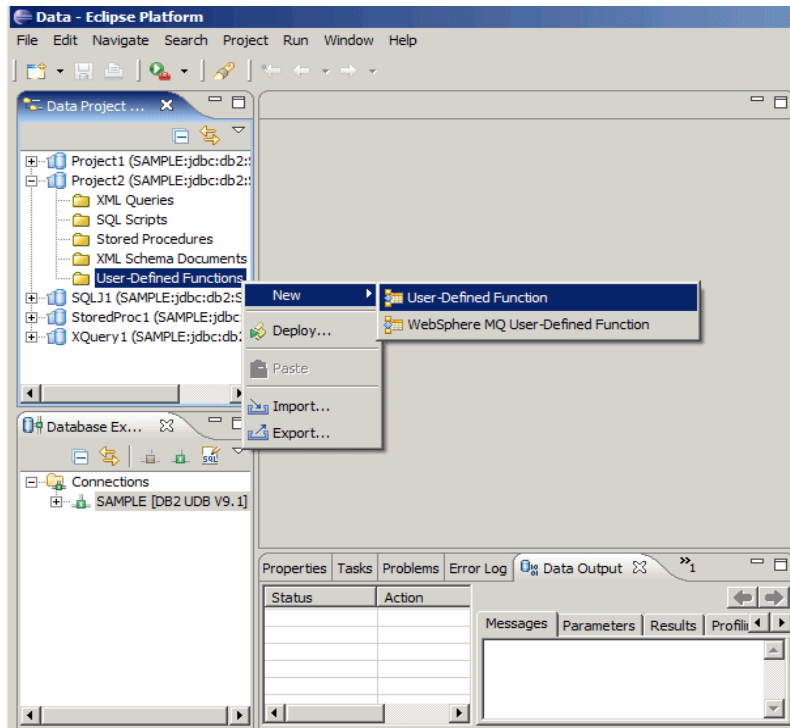


Figure 1-18 Selecting user defined function wizard

5. Complete the steps of the wizard.

Note: The New User Defined Function (UDF) wizard does not support all data types on all DB2 UDB servers. When creating a table UDF, the wizard recommends SQL data types for the data returned for each column.

By default, creating the user defined function does not register the user defined function on the database. To register the user defined function on the database, you must deploy it.

Creating XQuery using Visual Builder

DB2 9 allows you to write XQuery expressions directly rather than requiring that you embed or wrap XQueries in SQL statements. This is possible because the

DB2 9 query engine processes XQueries natively. We cover more details regarding XQuery in future chapters.

With native XML capabilities added to DB2 9, IBM has enhanced development tools for use with DB2 9. One of the key development tools is DB2 Developer Workbench, which utilizes XML Query Language (XQuery) to query the XML data type.

You can use Visual Builder to piece together queries, then review the DB2 Developer Workbench generated syntax.

The following walkthrough demonstrates the use of Visual Builder for the XQuery in DB2 Developer Workbench, which queries the name and city of the customer who has the city element defined with the data Toronto:

1. First launch DB2 Developer Workbench.
2. Establish a database connection.

In the Workspace panel, enter the workspace directory. The workspace is the Eclipse platform component that holds the development environment.

Select the **Database Explorer** tab in the workspace view (Figure 1-19 on page 34). Expand the Connections folder. This should show SAMPLE[DB Alias]. Right-click **SAMPLE[DB Alias]** then select **Reconnect**. The User ID and password prompt window appears. Upon successful connect, the status in the Properties should change from <Disconnected Connection>SAMPLE to <Live Connection> SAMPLE [DB2 UDB V9.1].

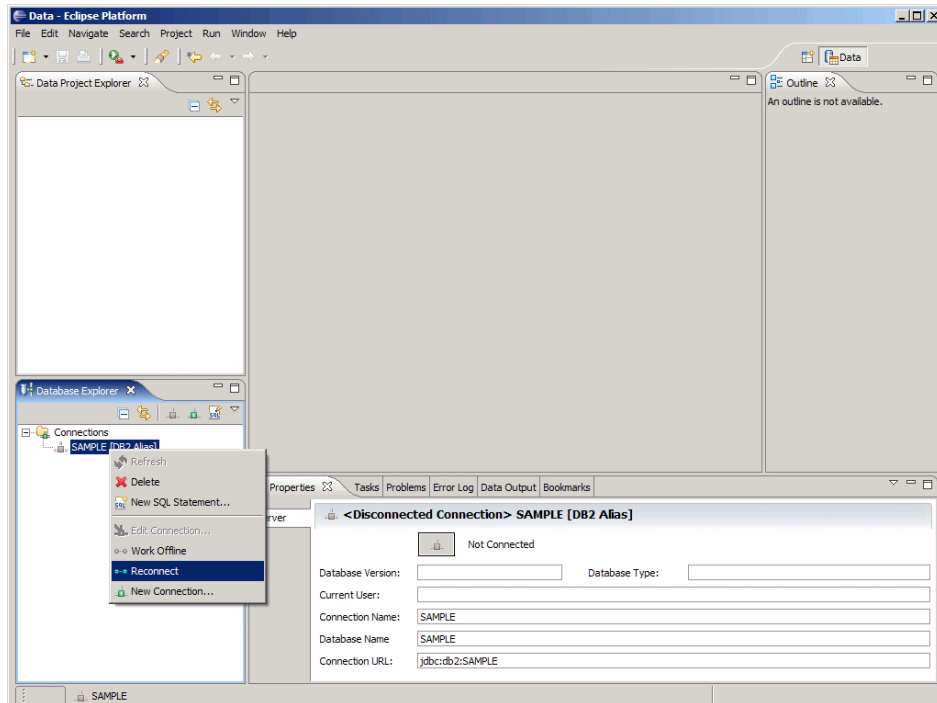



Figure 1-19 XQuery Visual Builder: connecting to database

3. Start a new data development project:

Use **File** → **New** → **Data Development Project**. You can make these same choices by clicking the **New Project** drop down  from the tool menu.

Name the project. In our example, XQuery1 was used as the project name. See Figure 1-20. Click **Next**.

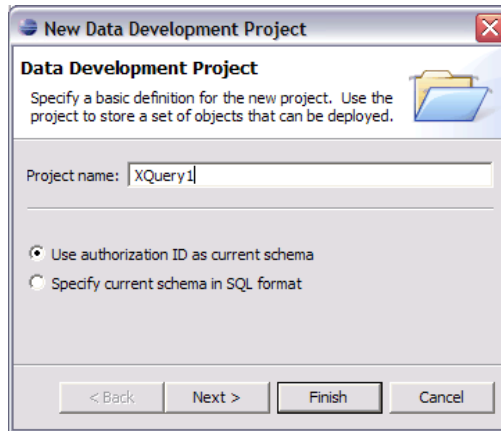


Figure 1-20 DWB: creating new project

4. In the Select Connection panel (Figure 1-21), you can choose either **Create a new connection** or **Use an existing connection**. Since connection has already been established in step 2 on page 33, we select **Use an existing connection** for our walkthrough. Click **Finish**.

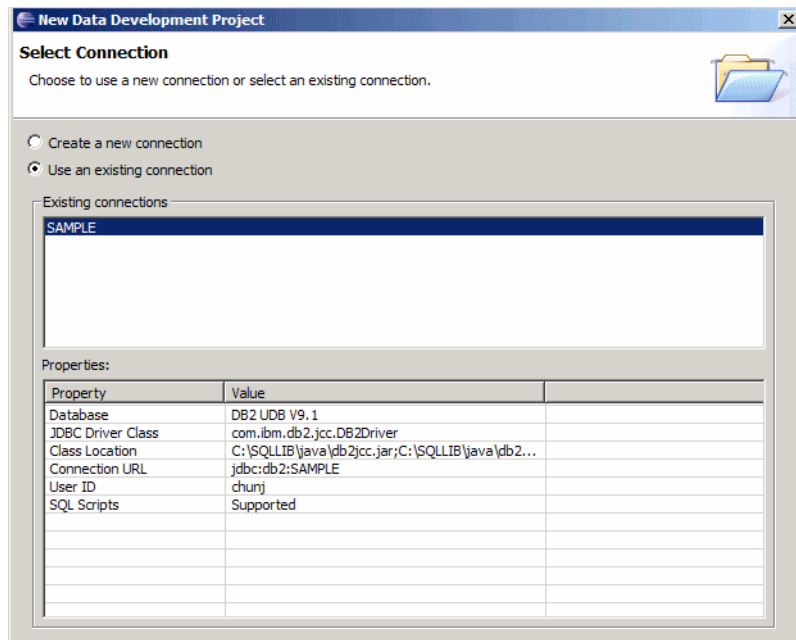


Figure 1-21 XQuery Visual Builder: specifying the connection

5. Create XML query.

Workbench can discover and generate the XML schema, but you can also add XSDs, DTD, and schema to the XML Schema Documents folder (This folder is located under the recently created project in the Data Project Explorer pane).

Right-click **XML Queries** and choose **New XML Query**. Name the query. Here we used `Sample_XML_Query`. Once you click **Next**, the Add representative XML documents window appears. Click the **Add** button on the right. On the Specify document location panel, choose **Database**. See Figure 1-22. Select **Next**.

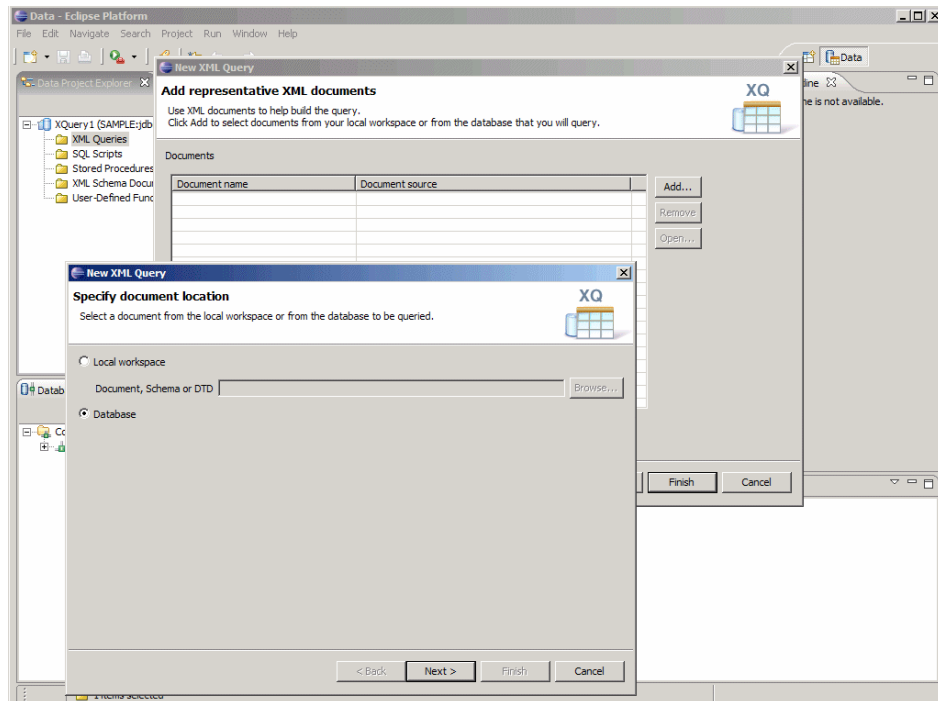


Figure 1-22 XQuery Visual Builder: specifying the document location

6. The XML column or Schema window displays (Figure 1-23 on page 37).

Select the schema, which contains the sample database, in this example, the schema is `CHUNJ`. We use the `CUSTOMER` table, which contains two XML data type columns (`INFO` and `HISTORY`). Choose the **INFO** column, and click **Next**.

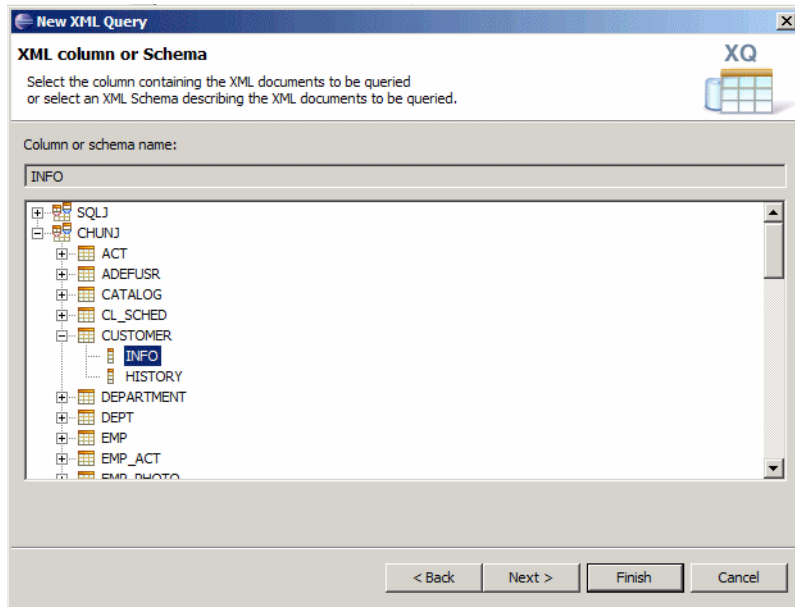


Figure 1-23 XQuery Visual Builder: specifying the document location

7. The Choose a document window shows you a list of XML documents found in that column as well as their respective sizes.

The Schema navigation window can be seen on the left side. Here you can specify the resulting XML file name (Document Name). In our walkthrough scenario, we accept the default. Click **Finish**. See Figure 1-24 on page 38.

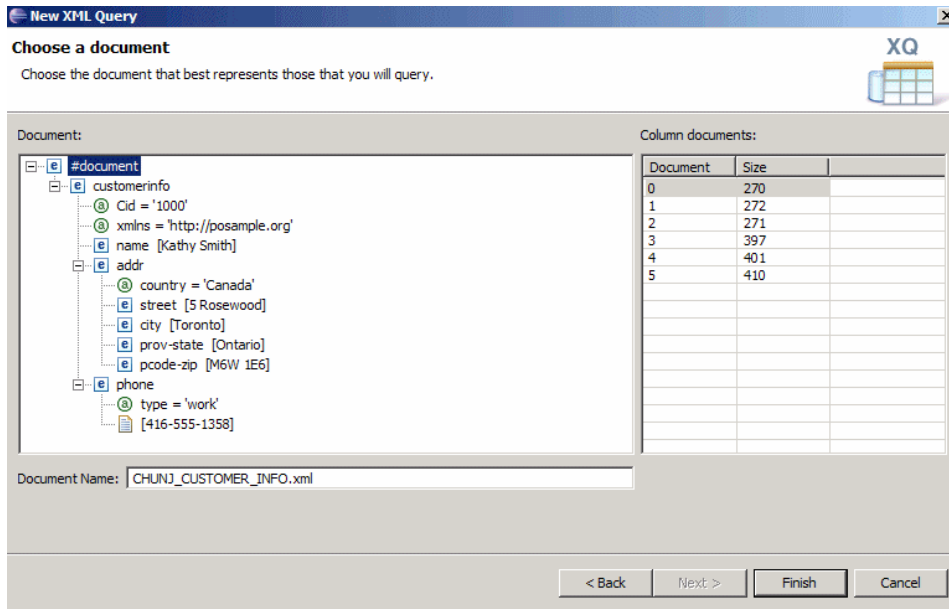


Figure 1-24 XQuery Visual Builder: specifying XML file name

8. The Add representative XML documents window displays. Now, we have successfully added data that we will query. See Figure 1-25. Click **Next**.

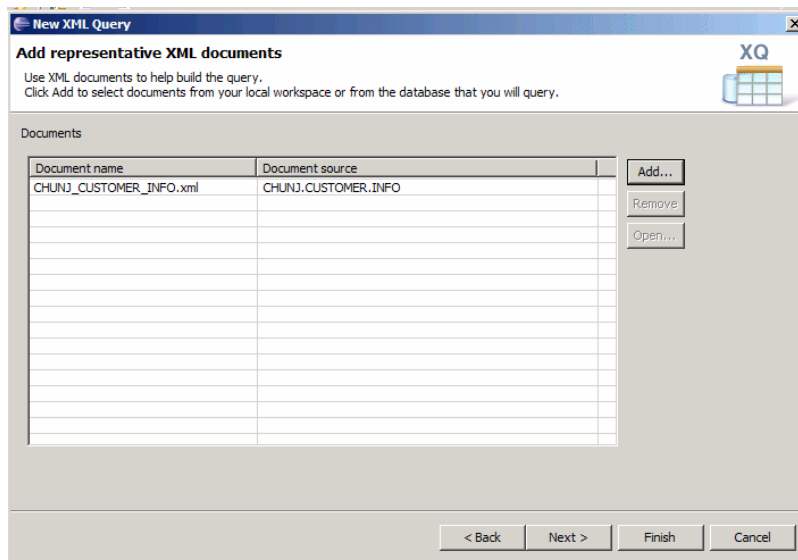


Figure 1-25 XQuery Visual Builder: adding representative XML documents

9. The Associate documents with XML columns window displays. This shows the association between XML Schema and actual data (shown by the green check mark in the **Associated?** column, see Figure 1-26). This association is there by default because the XML document was generated from the data in the specified XML column. The association between the XML document and the XML column is required for the query. Click **Finish**.

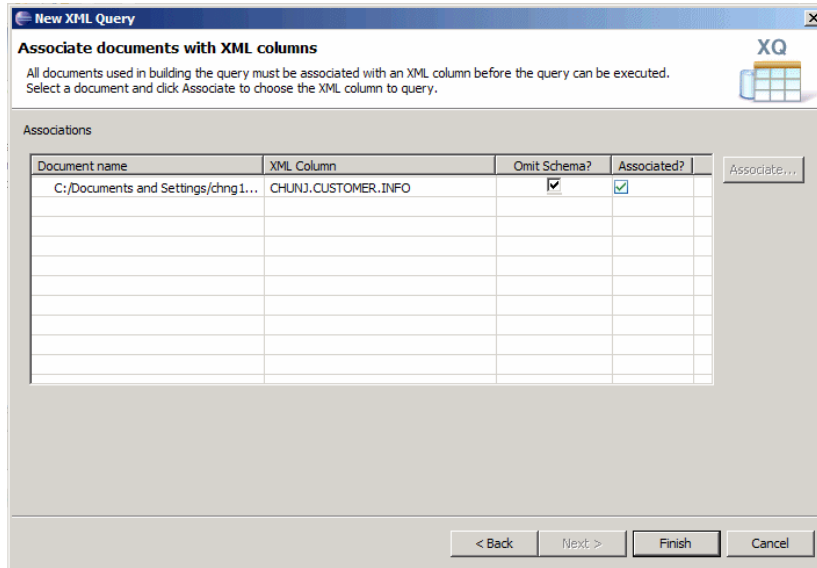


Figure 1-26 XQuery Visual Builder: associating documents with XML columns

10. This brings you to the workspace view with the Sample_XML_Query.xqm view. See Figure 1-27.

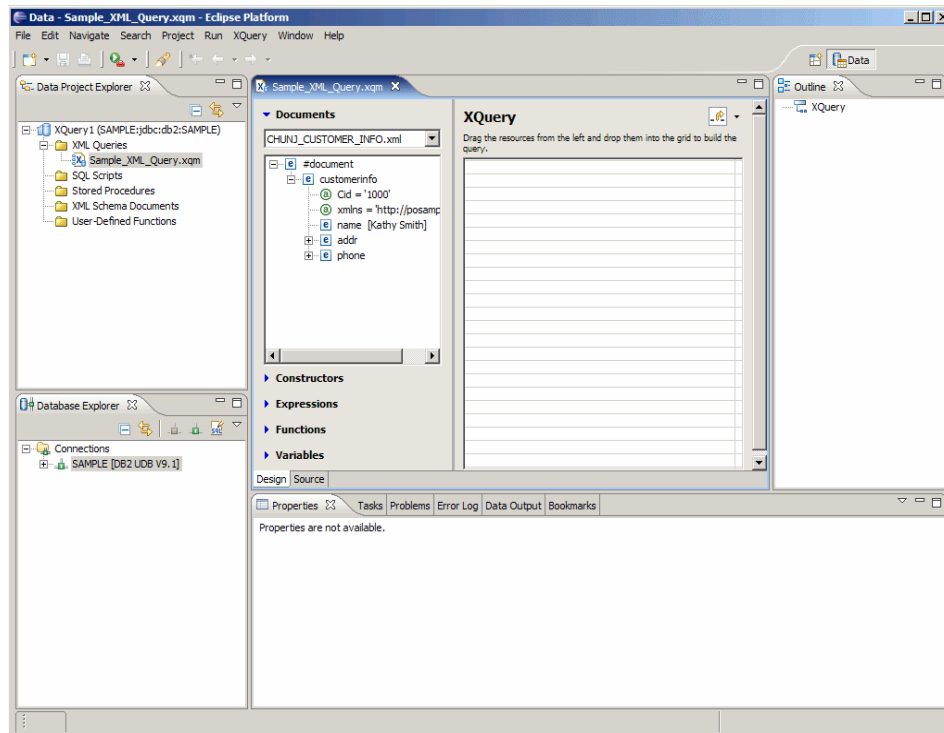


Figure 1-27 XQuery Visual Builder: XQM tab

11. The Sample_XML_Query.xqm view has a navigation node tree with Constructors, Expressions, Functions, and Variables syntax. We further explain this syntax in Chapter 2, “Application development with DB2 pureXML” on page 49.

Now drag the name node from the navigation tree over to the XQuery designer grid (right side of the XQM section). This node now serves as a representative node, which means that the query is performed for all XML data that has the name element. See Figure 1-28 on page 41.

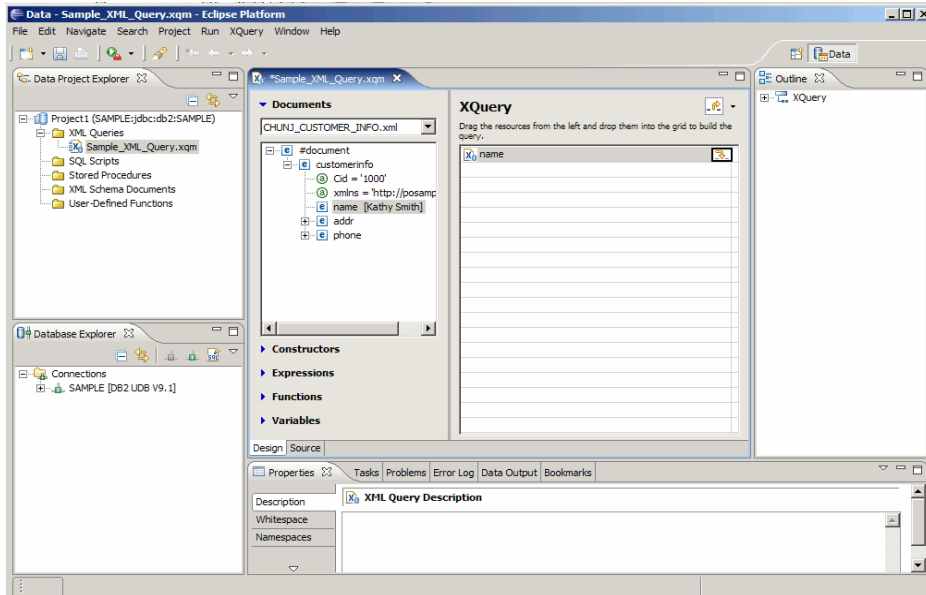



Figure 1-28 XQuery Visual Builder: adding element to XQuery

The Step Into icon  appears when name is highlighted. Use this to specify conditions and result sets for that node.

Click the **Step Into** icon to step into the For Logic (FLWOR) window. FLWOR stands for FOR, LET, WHERE, ORDER BY, and RETURN, which represent basic XQuery syntax.

We limit the query to cities with the name Toronto. So we expand the **Functions, String functions**, choose **fn:matches** and drag it to the Operand 1 column of the Where section. This populates the Operand 1 column in Where with matches(...) and the Step Into icon appears next to matches(...). See Figure 1-29. Click the **Step into** icon.

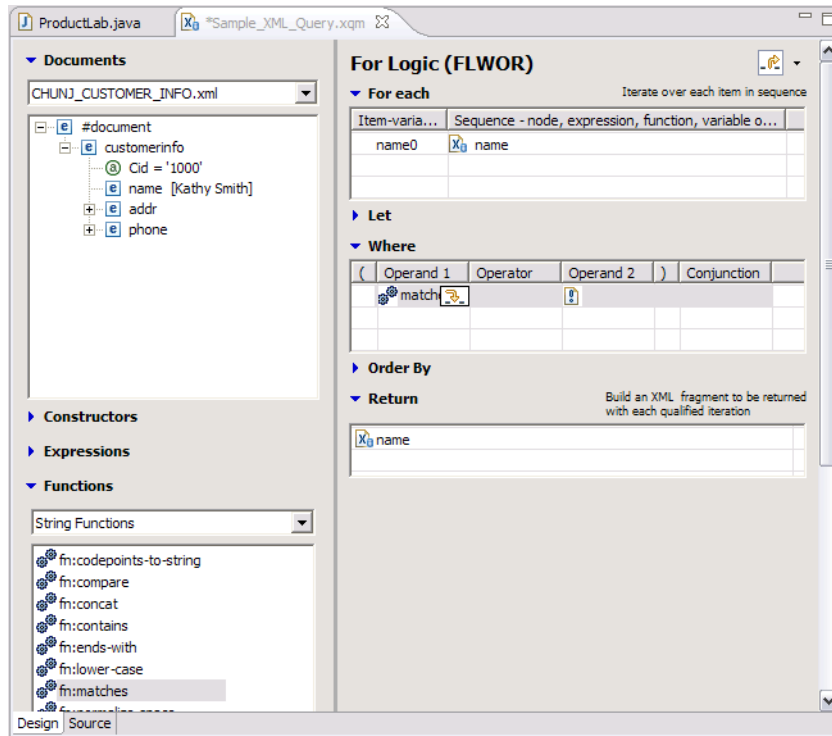



Figure 1-29 XQuery Visual Builder: For Logic (FLWOR) window

12. The matches window appears with arguments: source-string, pattern, and flags. Drag the city node under **addr** from the explorer window on the left to the source-string window. Enter Toronto for the pattern argument and i for case insensitivity for the flags argument. See Figure 1-30.

Now, click the **Step out** icon  on the top right corner of the matches window and return to the For Logic (FLWOR) window.

The fn:matches function does not take a second operand (Operand 2), so we leave that column empty (as well as Operator).

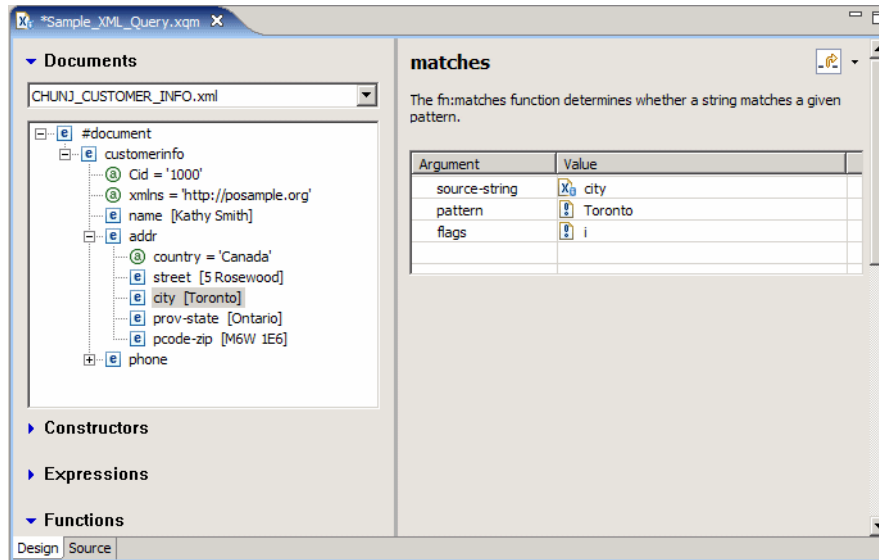


Figure 1-30 XQuery Visual Builder: matches window

13. In addition to the customer name of the customer who resides in Toronto, we want to display the city in the resulting output to verify that resulting names are those from Toronto.

To do this, drag the city node from the navigation pane on the left to the Return section of the For Logic (FLWOR) window. See Figure 1-31.

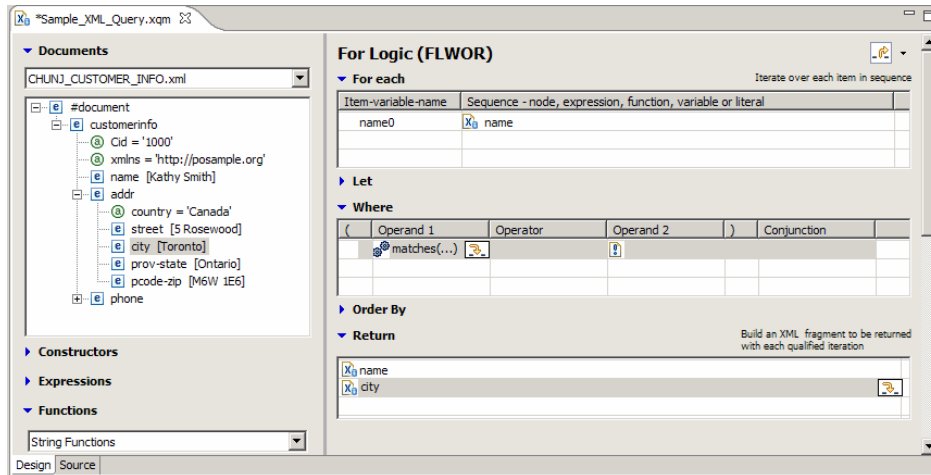


Figure 1-31 XQuery Visual Builder: For Logic (FLWOR) window with operand 1

14. There are two tabs, Design and source, representing two views in XQM window. We have been exposed to the Design view up to now. You can view the source generated by DB2 Developer Workbench by choosing the **Source** tab. See Figure 1-32.

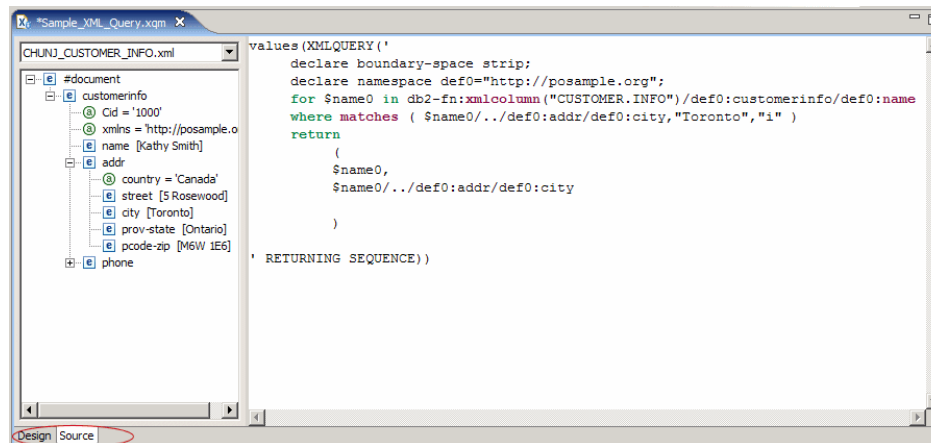


Figure 1-32 XQuery Visual Builder: source tab

You can execute the following source code that DB2 Developer Workbench generated (shown in Example 1-1 on page 45) in Command Editor with minor modification.

Example 1-1 Generated source code

```
values(XMLQUERY('
declare boundary-space strip;
declare namespace def0="http://posample.org";
for $name0 in
db2-fn:xmlcolumn("CUSTOMER.INFO")/def0:customerinfo/def0:name
    where matches ( $name0/../../def0:addr/def0:city,"Toronto","i" )
    return
        (
            $name0,
            $name0/../../def0:addr/def0:city
        )
' RETURNING SEQUENCE))
```

You can execute the same query from the script file as shown in Example 1-2.

Example 1-2 Query script

```
Values (XMLQUERY('
    declare boundary-space strip;
    declare namespace def0="http://posample.org";
    for $name0 in
db2-fn:xmlcolumn("CUSTOMER.INFO")/def0:customerinfo/def0:name where
matches ( $name0/../../def0:addr/def0:city,"Toronto","i" )
    return
        (
            $name0,
            $name0/../../def0:addr/def0:city
        )
,
RETURNING SEQUENCE))
@
```

Example 1-2 is the syntax for the XMLQuery. We can modify the source for XQuery as shown in Example 1-3.

Example 1-3 Modified source

```
xquery
    declare default element namespace "http://posample.org";
    for $name0 in
db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo/name where matches (
$name0/../../addr/city,"Toronto","i" )
    return
        (
```

```

$name0,
$name0/../../addr/city

)

@

```

Save the source in Example 1-3 on page 45 in a file (in this case, xquery.db2), then issue the following command:

```
db2 -td@ -vf xquery.db2
```

15. Now, execute XQuery using **XQuery** → **Run**. Figure 1-33 shows the successful execution of XQuery (see Data Output tab).

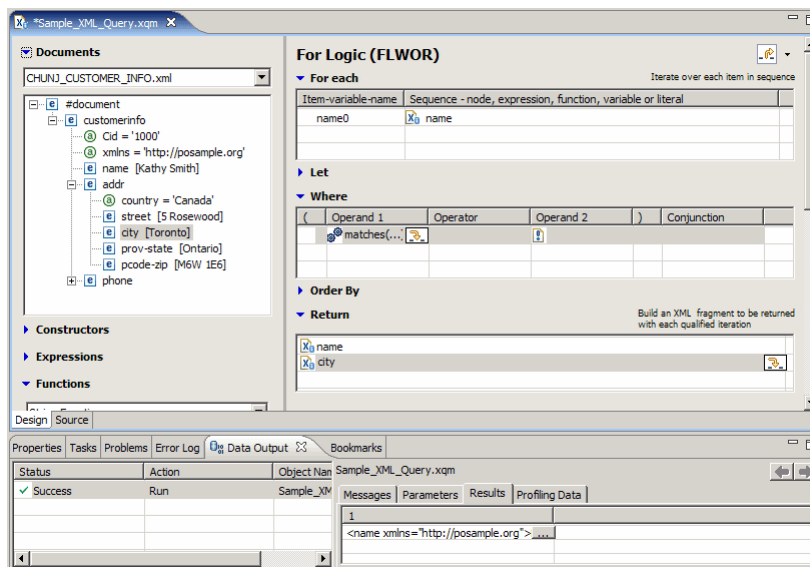



Figure 1-33 XQuery Visual Builder: Data Output tab

16. You can click the  button in the Results tab to see the results output. See Figure 1-34.

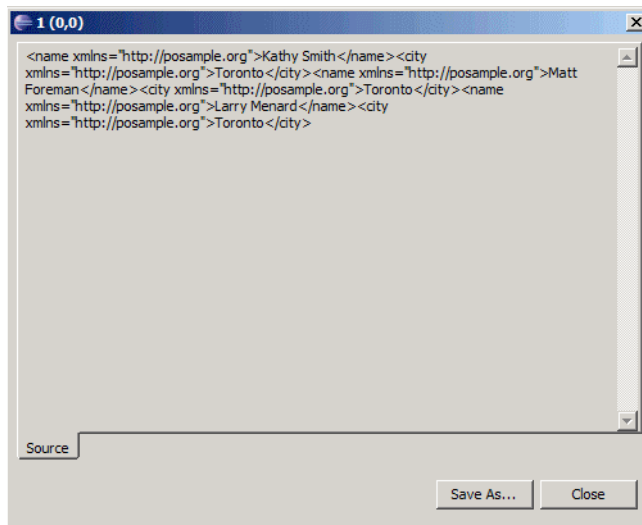


Figure 1-34 XQuery Visual Builder: XQuery results



Application development with DB2 pureXML

In this chapter, we discuss the technique of storing, managing, and querying XML data. XML is quite different from relational data, and it offers its own challenges and opportunities for the application developer. In this chapter, we introduce you to the new pureXML technology in DB2 Express-C V9. Since XML is everywhere, we have covered the application development topics for XML in each chapter of this book. We point you to the appropriate chapter and section for specific information throughout this chapter.

We provide practical examples to store and query XML data, which we encourage you to apply to your own environment. We provide the example XML data we use in this chapter in Appendix A, “Setup procedure and sample data” on page 295.

We also encourage you to visit the following DB2 XML Technical Papers and Articles Web site. Here you can find IBM white papers and links to the articles published in various publications on different topics related to the pureXML technology.

<http://www.ibm.com/developerworks/wikis/display/db2xml/Technical+Papers+and+Articles>

2.1 Web application: XML is the answer

Web applications are usually built by the following three tier model. These three tiers are considered separate well-defined processes or components that run on different platforms:

- ▶ The user interface or browser that runs on client machines
- ▶ The application programs that include the business logic runs on Web server
- ▶ The database management system or the back-end system that holds data

This application model has many advantages over the single or two tier model, because the application is accessed by the browser and shares the same look and feel, no matter where it is accessed from. Also, the application modularity makes it easy to modify one component without affecting others. However, a request to a Web application does not always need to originate from a human being. It also can originate from another application sharing the same model and same protocol. XML opens the door for connecting one Web application to another. Consider a Web-based traffic warning application, which gets the real-time traffic information from another Web-based application. A user subscribes to the traffic warning Web site, which processes the real-time traffic data and produces the warning result for its subscribers.

The architectural overview of such an application is shown in Figure 2-1 on page 51. If this Web application is developed using the old Hypertext Markup Language (HTML) technology, the traffic warning application would get an HTML data for one of its subscribers, who commutes between San Jose and San Francisco. The traffic warning application then would process the trouble spot as represented in a particular HTML tag on a particular line number (line#) of the HTML data. This would work fine for a very fixed format. However, needless to say, that application will break with a slight change in data format, or even if a blank line is inserted in the output generated by the real traffic application.

XML solves the problem; it directly represents the data. The real-time traffic data is returned as XML, which is the logical representation of the traffic data. The data representation is separate from page presentation. The fact that data is separate from presentation makes serving different types of contents to different types of client easier. The same traffic warning data can be accessed from a Web browser on your laptop as well as from a thin personal digital assistant (PDA) client on your cell phone.

Today, XML is the most common data interchange format on the Web. Any modern application can interact with an existing application through the Web

services technology, which exchanges information as messages represented in XML format. See Figure 2-1.

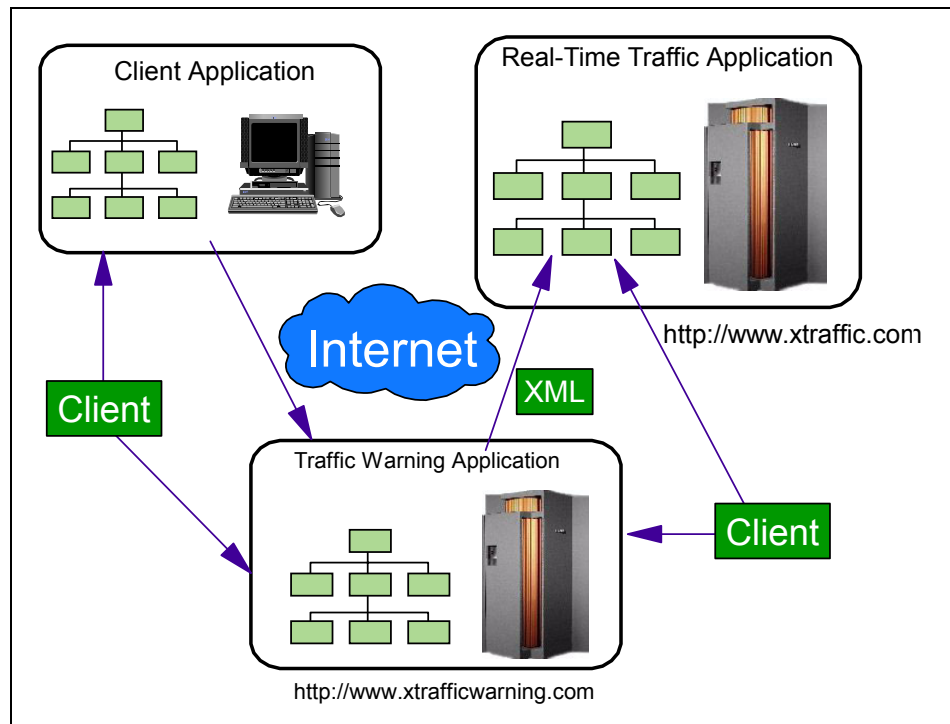


Figure 2-1 Web application XML: connecting each other

XML is so powerful and so flexible that people use it not only as a document but widely use it to:

- ▶ Describe the meta content of the Web resources. Resource Description Framework (RDF) is such an example, which is an XML representation of resource described uniquely on Web. The popularity of semantic Web technology is fueling the growth of rich metadata being defined on Web contents.
- ▶ Publish and interchange database contents. Sharing data between applications is a necessity. Finding a common ground and different parties agreeing to it is only possible through XML and XML schema.
- ▶ Communicate with message format. Business-to-business (B2B) applications use XML-based messaging to communicate.

So, why use XML? Well, for simplicity, richness for data structure, and its versatility (handling of international characters). Today, database vendors

support XML and hence your application can leverage the robustness and scalability of database systems while maintaining flexibility using XML data.

2.2 pureXML in DB2

pureXML is the brand new technology introduced in DB2 9. pureXML enables storage of well-formed XML documents in the DB2 column of XML data type. By storing XML data in XML columns, the data is kept in its native hierarchical form, rather than stored as text or mapped to a relational data model. Since the new XML store is 100% integrated with DB2, it leverages the performance, scalability, reliability, and availability of the DB2 engine.

The new XML data type is really a Structured Query Language (SQL) data type, so querying XML data is easy and fast. You can access relational and XML data in the same statement. The new pureXML is also integrated with application programming interfaces (APIs) such as Java Database Connectivity (JDBC), Open Database Connectivity (ODBC), .NET, embedded SQL, and PHP, which open doors for a new breed of Web applications with hybrid access to the relational and XML data. In this section, we discuss the application design considerations and benefits of pureXML technology in DB2 with respect to application development.

2.2.1 When to use DB2 pureXML

DB2 pureXML is suited for any application where some or all of the data processed by the application is represented using XML. DB2 pureXML storage provides for high performance ingestion, storage, processing, and retrieval of XML data. Additionally, it also provides the ability to quickly and easily generate XML from existing relational data.

The types of applications for which pureXML is particularly well suited include:

- ▶ Business-to-business and application-to-application (A2A) integration
- ▶ Internet applications
- ▶ Content-management applications
- ▶ Messaging-based applications
- ▶ Web Services

A typical XML-based application has one or more of the following requirements and characteristics:

- ▶ XML documents need to be processed or generated.
- ▶ High performance querying, both within a document and across a large collection of documents.

- ▶ High levels of security and easy control over them.
- ▶ Usage of languages such as Java that support open standards such as SQL, XML, XQuery, and Extensible Stylesheet Language Transformation (XSLT).
- ▶ Access to information using standard Internet protocols such as File Transfer Protocol (FTP), HTTP/Web, or JDBC.
- ▶ Hybrid access to and ability to query relational and XML data.
- ▶ Validation of XML documents.

2.2.2 Designing pureXML-based applications

This section discusses the preliminary design criteria you can consider when planning your DB2 pureXML-based application. Here are several questions to ask and our recommendations:

- ▶ Will your data be mostly XML, a combination of relational and XML, or non-XML?

Deciding how your data storage model will look mostly depends on the kind of application you are developing and the kind of data it will use. Sometimes, it might make more sense to store XML data in a relational table and query it using standard SQL if your XML data is highly regular and queried the same way over time. Often, you would want to store and represent data as XML throughout its life.

- ▶ Will your tables be XML schema-based or non-schema-based?

If your data is XML schema-based, you need to register the schema with DB2 XML schema repository, so that applications can use them to validate the data. You can experience performance issues if schema validation is used.

- ▶ How will other applications and users access your XML and other data?

The data access model really depends on the programming interface. DB2 supports the common application programming interfaces to access and manage XML data in pureXML.

- ▶ What kind of indexing will best suit your application? Will you need to use XML value index or full text search index? Or both?

You need to design indexes carefully for high performance data access. Text-based search or fuzzy search implementation needs to use full text indexes. This will depend on the nature of the application.

- ▶ In which languages will you program your application?

Java, .Net, and PHP are the most common languages a Web application uses. However, you might consider the latest community-developed

frameworks such as Ruby on Rails or Zend Framework to build Web applications that leverage persistent XML store in DB2.

- ▶ Will you need to generate XML from relational data or vice versa?
Your application can leverage XML constructor functions to build XML from relational data before applications can process it. In contrast, some use cases require XML data to be shredded into relation columns before the data can be accessed by an existing application.
- ▶ How often will XML documents be accessed, updated, and manipulated? Will you need to update fragments or the whole document?
Choosing the right query and interface model depends on whether an application is read only or both read and update. XQuery can limit your application to read only, so you need to carefully decide if full document update is reasonable.
- ▶ Will you need to transform the XML to HTML, WML, or other languages, and how will your application transform the XML?
If your application needs to be accessed by different clients, XSLT may be used to transform. While storing data in pureXML, you can make sure the structure complies with the specific XML schema.

2.2.3 DB2 hybrid query engine

DB2 hybrid engine processes SQL or XML and XQuery queries in an integrated manner. DB2 unifies XML storage, indexing, and query processing with existing relational storage, indexing, and query processing. DB2 also provides an XML schema repository (XSR) to register and maintain XML schemas and uses those schemas to validate XML documents. The DB2 utilities such as IMPORT and EXPORT have been enhanced to support XML data and the new graphical XQuery builder lets you construct XQueries.

The hybrid engine allows existing client applications to access DB2 data through relational APIs, yet it offers new SQL or XML APIs to publish relational data in XML format and full document retrieval from pureXML storage. Additionally, the new SQL or XML querying functions provide SQL applications with subdocument level search and extract capabilities by embedding XQuery statements into SQL statements.

An XML application can interact with the DB2 server via the XML interface by using the XQuery language, which is supported as a stand-alone query language independent of SQL. XQueries can optionally contain SQL statements to combine and correlate XML data with relational data. A client application can benefit immensely from this integration of the two languages supported by DB2.

DB2 has a separate parser for SQL and XQuery statements but uses a single integrated query compiler for both languages. No translation from XQuery to SQL is performed. DB2's compiler and optimizer are extended to handle SQL and XQuery in a single modelling framework. An overview of the hybrid engine is shown in Figure 2-2.

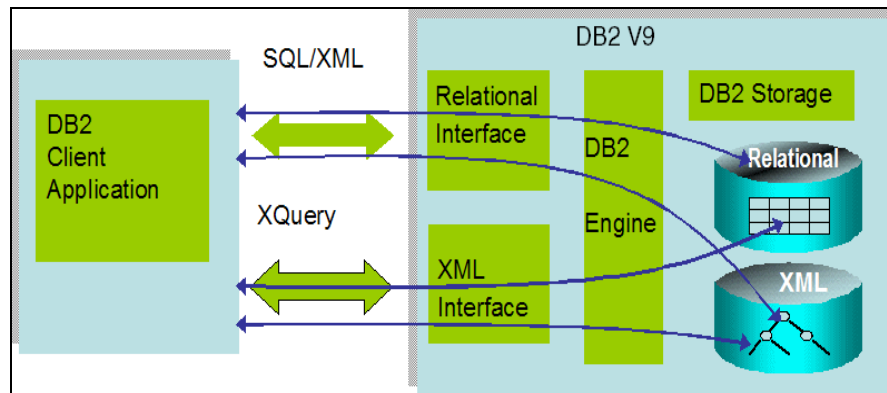


Figure 2-2 Integrating XML and relational data

2.2.4 pureXML storage overview

In DB2 pureXML, the XML documents are stored on disk pages in tree structures matching the XML data model. This avoids the mapping between XML and relational structures, and the impedance (hindrance) mismatch. XML is a data type in DB2, just like any other SQL type.

XML data type can be used in a CREATE TABLE statement to define one or more columns of type XML. Since XML has no different status than any other types, tables can contain any combination of XML and relational columns. A column of type XML can hold one well-formed XML document for every row of the table. The NULL value is used to indicate the absence of an XML document. Even though every XML document is logically associated with a row of a table, XML and relational columns are stored differently. XML schema is not required in order to define an XML column or to insert or query XML data. An XML column can hold schema-less documents as well as documents for many different or evolving XML schemas.

Schema validation is optional on a per document basis. Thus, the association between schemas and documents is per document and not per column, which provides maximum flexibility. Unlike a varchar or a character large object (CLOB) type, the XML type has no length associated with it. Currently, only the client-server communication protocol limits XML bind-in and bind-out to two GB

per document. With very few exceptions, this is acceptable for all XML applications.

You can use the XML type not only as a column type but also as a data type for host variables in languages such as C, Java, and COBOL. XML type is also allowed for parameters and variables in SQL stored procedures, user defined functions (UDFs), and externally stored procedures written in C and Java.

An application can manage XML documents in the best way suited for the application. These ways include shredding XML data into relational tables or just storing them as CLOBs. In DB2 pureXML, XML data is stored in a parsed, annotated tree form, similar to (but distinct from) the Document Object Model (DOM). The XML data is formatted to data pages, which are buffered. The benefits of this format include faster navigation, which results in faster query execution as well as simpler indexing of data. Figure 2-3 shows a DB2 CREATE TABLE statement with XML data type, and how relational and XML data is stored in DB2.

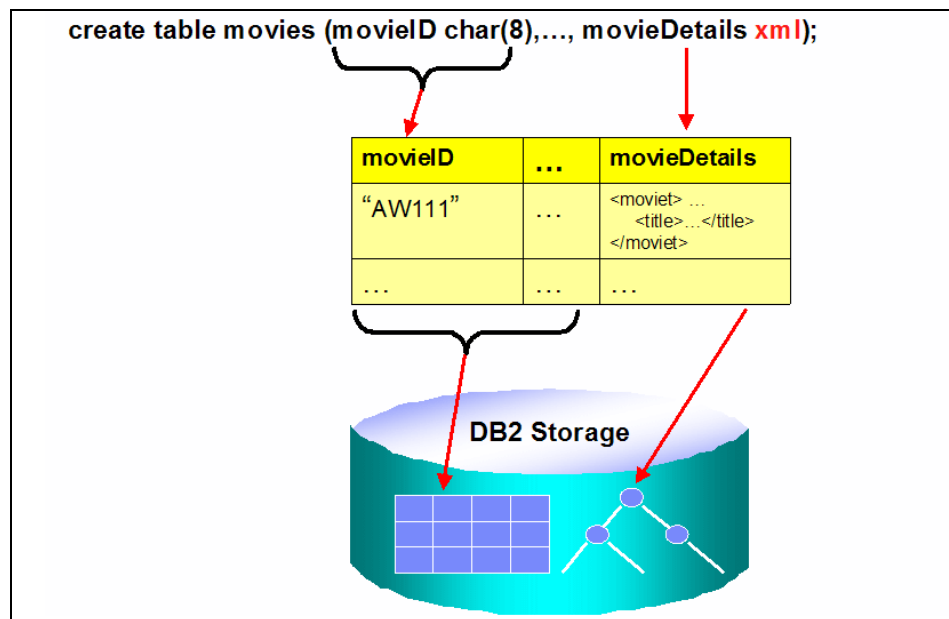


Figure 2-3 Creating table with XML data type

Throughout this chapter, we have used examples of XML data, relational data, XQueries, SQL queries, and so on. These examples are based on Movie data. The examples might not be entirely consistent because they were produced to illustrate specific points made in the text and they might not be always complete. The actual XML data and steps to create them in your environment have been

discussed in detail in Appendix A, "Setup procedure and sample data" on page 295. In summary, we have used two database tables for our example as shown in Table 2-1 and Table 2-2.

Table 2-1 Movie database table

ID	int NOT NULL PRIMARY KEY
INFO	XML

Table 2-2 Moviereview database table

REVIEWID	int NOT NULL PRIMARY KEY
REVIEW	XML

The sample XML data for Movie is shown in Example 2-1.

Example 2-1 Movie.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<movie id = "345">
  <heading>
    <title>Crossroads</title>
    <rating>***</rating>
  </heading>
  <movie-details>
    <genres>Drama, Romance and Remake</genres>
    <MPAArating>PG</MPAArating>
    <country>US</country>
    <year>2006</year>
    <running-time>1 hr. 38 minutes</running-time>
    <production>
      <studio>RR Pictures</studio>
    </production>
  </movie-details>
  <synopsis>
    A grouchy old college professor who lived in a delapidated cabin near
    the lake takes an interest in a frustrated writer who suffers from
    habitual writer's block.
  </synopsis>
  <credits>
    <writer>David Anderson</writer>
    <director>Alex Ostern</director>
    <photography>Charles Govasky</photography>
    <actors>
      <actor special="yes" type="lead" gender="M">Karl Thomas</actor>
      <actor type="lead" gender="F">Sandra Casper</actor>
      <actor type="lead" gender="M" special="yes">Peter Walsh</actor>
    </actors>
  </credits>
</movie>

```

```
        <actor gender="M">Sam Allen</actor>
        <actor gender="M">Chris Plummer</actor>
        <actor gender="F">Linda Collins</actor>
        <actor gender="M">Bill Hagen</actor>
    </actors>
</credits>
</movie>'
```

The sample data for Movie review is shown in Example 2-2.

Example 2-2 Moviereview.xml

```
<movie id = "345">
  <reviews>
    <UserReview>
      <user name ="Andy">Three is a crowd, yet there are not enough stars
        to support this catastrophe.</user>
      <user name ="Linda">I really liked this movie.</user>
    </UserReview>
    <CriticsReview>
      <newspaper name="ABC Times">The best mystery thriller of the year.
        </newspaper>
      <newspaper name="San Francisco Post">Its ok...movie.
        </newspaper>
    </CriticsReview>
  </reviews>
</movie>
```

You can find the complete data for Example 2-1 on page 57 and Example 2-2 in Appendix A, "Setup procedure and sample data" on page 295.

All the example queries in this chapter have been tested against the movie and moviereview data in the DB2 9 database. We advise you to follow the setup steps mentioned in the Appendix A, "Setup procedure and sample data" on page 295 to get the desired result from these example queries.

2.2.5 SQL support for XML data (INSERT, SELECT)

In DB2 9, XML is a new supported data type. In the CREATE TABLE statement, the XML data type can be specified as a column data type. Just like other data types, while inserting an XML value in the column of type XML, DB2 makes sure that the data type of a value is XML.

XML value is made of nodes of different data types and can be represented and stored in any of the character types, such as CHAR, VARCHAR, CLOB, BLOB,

and so on. Before inserting a value in the XML columns, DB2 makes sure that the string representation contains a valid XML document.

Similarly, when selecting the XML value, it should be converted to any of the character data types so that it can be presented or used in the application. DB2 provides functions to explicitly check the correctness of the XML value before inserting and converting the XML value to a string representation while selecting.

An XML document can have the encoding information mentioned in the document itself in the declaration section. Such documents are called *internally encoded documents*. The encoding used for the documents, which do not contain this information, depends on the application code page.

XMLPARSE

You can optionally use the *XMLPARSE* function to explicitly check the correctness of the XML document in an SQL statement. The input to this function can be any of the character types: CHAR, VARCHAR, and CLOB, or BLOB. The function returns the parsed XML value if it is a valid XML document; otherwise, it throws an error.

As only the valid XML values are allowed in an XML column, the XMLPARSE function is mandatory to use when inserting the XML value either externally or internally. If the XMLPARSE function is not used in an insert statement explicitly, it is used implicitly to make sure that the value is correct XML. This type of parsing is called *implicit parsing*.

Whenever a character type is passed to the XMLPARSE function, it has an external encoding associated with it. It can be possible that the XML value stored in the character type variable is also internally encoded. The internal code page of the XML value in the character type variable and the external code page associated with the variable should match. If these code pages do not match, a run-time error is thrown. We recommend you not use the encoding attribute whenever a character type is used to represent XML data. For binary data types, there is no code page associated with the type, the XML value is treated as encoded in the internal code page specified in the XML declaration. In case there is not internal encoding associated with the value, XML is treated as unicode.

Tip: Implicit parsing is the recommended way to work with XML data because it takes care of any encoding conversion in the best possible manner.

Example 2-3 gives an example of an INSERT statement where XMLPARSE is used explicitly to make sure that the value is well formed XML.

Example 2-3 XMLParse function

```
insert into moviereview(reviewid, review) values(765, xmlparse(
document '<movie id = "567">
  <reviews>
    <UserReview>
      <user name ="Andy">Three is a crowd, yet there is not
        enough stars to support this catastrophe.</user>
      <user name ="Linda">Three is a crowd, yet there is not enough
        stars to support this catastrophe.</user>
    </UserReview>
    <CriticsReview>
      <newspaper name="ABC Times">The best mystery thriller
        of the year. </newspaper>
      <newspaper name="San Francisco Post">The best mystery thriller
        of the year. </newspaper>
    </CriticsReview>
  </reviews>
</movie>'))
```

A host variable or parameter of type XML cannot be given as input to the XMLPARSE function. They are implicitly parsed by the database.

XMLPARSE does not perform validation against a schema and only provides the parsing functionality to check the correctness of the XML value and how well-formed the XML value is.

The full syntax for the XMLPARSE is:

```
XMLPARSE (DOCUMENT <String Value> PRESERVE/STRIP WHITESPACE
```

XMLPARSE provides two options for extra whitespace processing while parsing the string value:

- ▶ PRESERVE WHITESPACE: Preserve the whitespace in the string value.
- ▶ STRIP WHITESPACE: Remove the extra whitespace. This is the default action if no option is provided.

The DOCUMENT keyword specifies that the string value is a well-formed XML document which conforms to XML Version 1.0.

XMLVALIDATE

You can use the *XMLVALIDATE* function to validate the XML value against a schema. The schema for an XML value defines the structure of the XML values including node data types, occurrences, default values, and so on. The schema document should be registered to the database before using it for the validation.

Example 2-4 gives an example of the XMLVALIDATE function. This example assumes that the schema is registered with the name *review*.

Read 2.2.6, “Schema support” on page 62 to see how to register the schema used in Example 2-4.

Example 2-4 XMLVALIDATE function

```
insert into moviereview(reviewid, review) values(765,
xmlvalidate(xmlparse( document '<movie id = "567"
xmlns="http://movies.org">
  <reviews>
    <UserReview>
      <user name ="Andy">Three is a crowd, yet there are not
        enough stars to support this catastrophe.</user>
      <user name ="Linda">Three is a crowd, yet there are not
        enough stars to support this catastrophe.</user>
    <UserReview>
    <CriticsReview>
      <newspaper name="ABC Times">The best mystery thriller
        of the year. </newspaper>
      <newspaper name="San Francisco Post">The best mystery thriller
        of the year. </newspaper>
    </CriticsReview>
  </reviews>
</movie>') according to xmlschema id review));
```

XMLVALIDATE takes a well-formed XML value as an input. If a value of character type is passed to this function, the value is implicitly parsed before validation.

XMLSERIALIZE

When selecting an XML column from the database table, it should be serialized to CHAR, VARCHAR, CLOB, or BLOB data types so that the value can be used by the application. Use the *XMLSERIALIZE* function for this same purpose. It serializes an XML value to its textual representation. XMLSERIALIZE takes an XML value or XML column name as input in an SQL statement.

When serializing the XML value, the code page associated with the serialized data depends on the target data type used. For character data types, the encoding is application code page. For binary data types, the encoding associated with the serialized data is UTF-8.

Example 2-5 gives an example of how to use the XMLSERIALIZE function when selecting an XML column.

```
select xmlserialize(review as varchar(500)) from moviereview
```

If the SERIALIZE function is not used when selecting an XML column, the XML value is serialized to BLOB value implicitly.

2.2.6 Schema support

XML schema defines the structure of the XML document, including the element definition, attribute definition, namespaces, code pages, and so on. These schemas can be used to validate the XML value before inserting these values into the database table. Generally these schemas are stored in the file system and need to be registered to the database before using them for validation. Registering a schema to the database makes the schema a database entity and removes the dependency of reading the content of the schema document from the operating system. DB2 provides two ways of registering the schema to the database.

Registration using CLP commands

DB2 introduced new CLP commands to register a schema to the database. To register the database using the CLP commands, follow these steps:

1. Register the primary schema document.

The primary schema document is the one at the top of the import/include hierarchy. Register the main schema document using the following command:

```
register xmlschema <namespace URI> from <location URI> with  
<property URI> as <relational id>
```

The terms in the command are as follows:

- Namespace Uniform Resource Identifier (URI): Namespace of the XML document which will be referenced from XML documents.
- Location URI: Physical location of the schema file in the file system.
- Property URI: Physical location of the property document associated with the schema document in the file system.

2. Add the secondary schemas document.

Add the XML schema document referenced (using the include or import tag) in the main schema document or any of the already added secondary schema documents:

```
add xmlschema document to <relation id of main schema> add <document  
URI> from <location URI> with <property URI>
```

3. Complete the registration using:

complete `xmlschema` *<relational id of main schema>* with *<property URI>* enable decomposition.

The **complete XMLschema** command throws an error in the case where all the schema documents referenced in the primary or secondary schema documents are not added to the main schema. For the *include* tag, the namespace of the primary schema and the included schema should be the same. For the *import* tag, it is different and should be mentioned in the **add xmlschema** command.

All the three steps can be combined in one command, **register xmlschema**, by adding the **add** and **complete** clause.

Example 2-6 gives the schema example for the REVIEW column in the MOVIEREVIEW table.

Example 2-6 Schema for review column

```
<?xml version="1.0"?>
<xs:schema targetNamespace="http://movies.org"
xmlns:xd="http://movies.org"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:complexType name="userreview">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="userreviewtype">
    <xs:sequence>
      <xs:element name="user" type="xd:userreview" minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="criticsreviewtype">
    <xs:sequence>
      <xs:element name="newspaper" type="xd:userreview"
minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="reviewtype">
```

```

    <xs:sequence>
    <xs:element name="UserReview" type="xd:userreviewtype"/>
    <xs:element name="CriticsReview" type="xd:criticsreviewtype"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="movietype">
  <xs:sequence>
    <xs:element name="reviews" type="xd:reviewtype"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:integer"/>
</xs:complexType>

<xs:element name="movie" type="xd:movietype"/>
</xs:schema>

```

Example 2-7 gives the command to register this schema to the database. Because we are not importing any schema within the main schema shown in Example 2-6 on page 63, there is no need to use the **add xmlschema** CLP command.

Example 2-7 Registering schema

```

register xmlschema http://movies.org from review.xsd as review
complete xmlschema review

```

Registration using stored procedures

You cannot use CLP commands in the higher programming languages. To register an XML schema using an application program in one of the higher programming languages, DB2 provides three stored procedures equivalent to the three CLP commands:

► SYSPROC.XSR_REGISTER

This stored procedure can be used to register the primary XML schema:

```
XSR_REGISTER(rschema, name, schemalocation, content, docproperty)
```

The terms in the command are explained as follows:

- **rschema**: Relational schema name for the XML schema. This parameter is of the type VARCHAR(128). NULL value for this argument specifies the CURRENT SCHEMA.
- **name**: Relational identifier for the XML schema. This parameter is of the type VARCHAR(128). The fully qualified name for the schema ID becomes rschema.name.

- schemalocation: Namespace of the primary XML schema. This parameter is of the type VARCHAR(1000).
 - content: Content of the XML schema document. This parameter is of the type BLOB(30M). This is a non-nullable parameter.
 - docproperty: Content of the property document associated with the schema. This parameter is of the type BLOB(5M).
- **SYSPROC.XSR_ADDSCHEMADOC**
- Use this stored procedure to add the XML Schema documents to the primary schema:
- ```
XSR_ADDSCHEMADOC (rschema, name, schemalocation, content,
docproperty)
```
- The explanations for the terms in the command are:
- rschema: Relational schema for XML schema
  - name: Relational Schema name for primary XML schema
  - schemalocation: Namespace for the XML Document
  - content: Content of the XML document as a BLOB variable
  - docproperty: Property document associated with the XML schema document
- **SYSPROC.XSR\_COMPLETE**
- Use this procedure to complete the registration:
- ```
XSR_COMPLETE(rschema, name, schemaproperty, isusedforshred)
```
- The terms in the command are explained as follows:
- rschema: Relational schema for the XML schema.
 - name: Relational identifier of the primary XML schema.
 - schemaproperty: An input argument of type BLOB (5M) that specifies properties, if any, associated with the XML schema. The value for this argument is either NULL, if there are no associated properties, or an XML document representing the properties for the XML schema.
 - isusedforshred: The value for this parameter is `true` if the schema is used for decomposition, otherwise, the value is `false`.

2.2.7 Annotated XML schema decomposition

XML decomposition or *shredding* is the process of breaking down an XML document into columns of relational tables. Consider that your organization can have an existing relational database repository and business applications built on

top of this data, and you want to capture new data coming as XML and use it with existing applications. Consider another case of a need to shred XML documents conforming to different XML schemas in a single relational table. Using the new XML decomposition feature of DB2 Express-C V9 is the right choice for both of these situations.

XML decomposition

XML decomposition can be helpful because:

- ▶ It can break one large XML document into fragments that can be stored in column types such as XML, CLOB, VARCHAR, BLOB, and so on.
- ▶ It can shred multiple items into the same tables-column pair.
- ▶ It can automatically detect one-to-many relationships between columns mapped to the same table.
- ▶ It performs type conversions from an XML schema type to DB2 type.
- ▶ It can validate during shredding.
- ▶ It can specify custom expression on contents before insertion.
- ▶ It can specify path conditions, for example, shred ** only if the element is in path */a/b*.
- ▶ It can specify conditions on tuples.

Performing decomposition

To decompose an XML data, follow these steps:

1. Create the tables (relational schema) into which you decompose the XML data.
2. Annotate your XML Schema with mapping information:
 - Annotate manually.
 - Use DB2 Developer Workbench.
 - Migrate existing DADs to Annotated XML Schema using tool.
3. Register it in the XML Schema Repository via a CLP command.

```
REGISTER XMLSCHEMA 'http://myOrderSchema/order.xsd' FROM
'file://c:/temp/order.xsd' AS user1.myOrderSchema COMPLETE ENABLE
DECOMPOSITION
```
4. Register via the stored procedure.
 - a. Set the value of the “isusedforshred” to true.
5. Register via the JDBC function.
6. Decompose XML documents via the CLP command.

```
DECOMPOSE XML DOCUMENT c:\mydoc.xml XMLSCHEMA john.myschema
```

7. Decompose documents by calling the stored procedure.

Choose SP depending on the size of the document to be shredded:

< 1 MB, < 10 MB, <25 MB, 50 MB, <75 MB, < 100 MB

XDBDECOMPXML

XDBDECOMPXML10MB

XDBDECOMPXML25MB

XDBDECOMPXML50MB

XDBDECOMPXML75MB

XDBDECOMPXML100MB

Tip: The tables and columns must preexist before the annotated XML schema is enabled for decomposition. All missing objects (tables or columns) to which the XML schema refers are reported as errors. Also the CLP command automatically calls the correctly sized stored procedure depending on the size of the XML document. If a stored procedure with 100 MB is called for documents smaller than 1 MB, then the application's performance might be adversely affected due to high memory consumption.

We recommend the new Annotated schema decomposition feature of DB2 9 over the old XML Extender shipped with previous releases of DB2 because the new technique is much faster if XML schema is used, and it has more mapping constructs and other features. However, if you use old XML Extender to map and shred documents into DB2, you might be interested in a migration document published on the IBM developerWorks® Web site:

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0604pradhan/>

This article also provides DAD mapping techniques and a schema conversion utility. The DAD to annotated XML schema converter utility helps users convert their XML schemas to annotated XML schemas based on the mapping rules described in the DAD. The XML schema and DAD must describe the same set of XML documents, although it is possible and acceptable that the XML schema can describe a super set of documents described by the DAD. Users who do not have an XML schema can easily generate XML schemas from corresponding DTDs and even XML documents by using tools available freely on the Internet. The DAD to annotated XML schema converter tool can then take the XML schema and the DAD as input to produce an annotated XML schema.

The tool supports import, include, and redefine construct of the XML schema. In other words, if an XML schema is spread across many XML schema documents through import, include, or redefine, only the path for the primary schema document, the document through which all XML schema documents can be

reached through either import, include, or redefine, is needed. The tool annotates element or attribute declarations across schema documents. Note that since DADs do not support namespaces, it is impossible to have the use of an import construct in this scenario.

For more information about different annotation techniques for the new XML decomposition, refer to *DB2 XML Guide*, SC10-4254.

2.2.8 XML query support

For querying XML values, DB2 supports XQuery language, which you can use in conjunction with SQL to retrieve data from the XML columns. So, along with the SQL interface, DB2 provides another interface, XQuery, to query the XML values. These interfaces interact with each other using SQL or XML functions. Using these functions, XQuery can be embedded into an SQL statement to fetch the part of the XML document from each row of the SQL statement result.

To apply the pure XQuery on a sequence of XML documents fetched from the database, DB2 provides resource functions to fetch the XML values.

This section discusses the following:

- ▶ Functions used to embed SQL statement in an XQuery (XML source functions for XQuery)
- ▶ Writing XQuery
- ▶ Function used to embed XQuery statements in SQL (SQL or XML functions)

XML source function

DB2 XQuery can be applied either on the data stored in the XML column of the tables or an XML-created document using the constructor functions. For applying XQuery on XML column data, XML values must be fetched from the column to create a sequence of XML values. DB2 provides the following function to create a sequence of XML values by fetching the data from XML columns.

xmlcolumn

The *xmlcolumn* function takes the qualified name of the XML column and returns the sequence of XML documents stored in the XML column.

The syntax for the *xmlcolumn* function is:

```
db2-fn:xmlcolumn('<schema name>.<table name>.<column name>');
```

Default value for the schema is the current schema.

Example 2-8 gives the example for an `xmlcolumn` applied on the `info` column of the `movie` table.

Example 2-8 xmlcolumn applied on info column of the movie table

```
xquery db2-fn:xmlcolumn('MOVIES.INFO')
```

Tip: DB2 is case insensitive and treats all the table and column names in capital letters while XML and XQuery are case sensitive. The resource functions previously discussed are XQuery interface functions so all the table names and column names should be passed to these functions using capital characters or letters. Passing the object names in lower case letters can result in an undefined object name error.

sqlquery

Instead of querying all the XML values in an XML column (as in the case of `xmlcolumn` functions), we can only query a set of XML values based on a condition in a `SELECT` statement using the `sqlquery` function, for example, querying only the information regarding the movie with ID value 123. For this purpose, the `sqlquery` function provides you with the option to give the SQL full select as an input to the function instead of just the column name. If you give the full select as an input to this function, you should select XML values only. The function returns the concatenation of all the values selected by the full select.

Example 2-9 shows how to use the `sqlquery` functions to select the XML values.

Example 2-9 sqlquery

```
xquery db2-fn:sqlquery('select info from movies where id=123')
```

Tip: `xmlcolumn` and `sqlquery` are xquery interface functions and are case sensitive. Trying to use these functions in upper case letters results in an error.

Writing XQuery

The XQuery language queries the XML data. XQuery works on a sequence of XML documents generated by either using the `resource.functions.constructor` function or created as an intermediate result of another XQuery. Use XQuery, along with SQL/XML functions, to query the XML documents stored in the DB2 9 database.

Paths and predicates

XQuery queries the XML data based on the path of the nodes in the XML hierarchical structure. The nodes in the XML document relate to each other. The relationships that are supported between two nodes are:

- ▶ Child
- ▶ Attribute
- ▶ Descendant
- ▶ Parent
- ▶ Self
- ▶ Descendant or self

You can query an XML node based on these relationships by providing the path of the node in the XML value.

Example 2-10 shows a small query, which select all movies from the MOVIES.INFO table. A path based on the previous relationship is used to reach the title node in the XML value.

Example 2-10 Axis and path in XQuery

```
xquery db2-fn:xmlcolumn('MOVIES.INFO')/movie/heading/title
```

The forward slash (/) in the query indicates the next step in the path followed by the axis (relationship) to move and the node name.

The query in Example 2-10 uses the relationship name to indicate the next step or direction in the path. This format is called the *abbreviated version*. There is also the unabbreviated format, which uses the name of the relationship to indicate the next step. Table 2-3 compares the syntax between the unabbreviated and abbreviated formats. The abbreviated format that is defined for each relationship reduces the size of the query and is the commonly used format.

Table 2-3 Abbreviated and unabbreviated syntax

Abbreviated syntax	Unabbreviated syntax
dept /dept/emp	child::dept /child::dept/child::emp
/@id /department/emp/@id	/attribute::id /child::department/child::emp/attribute::id
//name /child::dept//name	/descendant::name /child::dept/descendant::name

So the default axis is the child axis. The attribute axis can be represented by the at sign (@). Similarly, the descendant axis can be represented by two forward slashes (//).

You can filter the values selected by the path in the Query by a condition in the same way we filter the value using the WHERE clause of the SELECT statement. You do this by using the predicate in XQuery. Let us select all the US-based movies from our movies database. Example 2-11 gives you the query to get the required result.

Example 2-11 Predicate in XQuery

```
xquery
db2-fn:xmlcolumn('MOVIES.INFO')/movie[movie-details/country='US']/heading/title
```

The predicate in an XPath starts with a bracket ([) and ends with the closing bracket (]). A predicate is evaluated for the given condition and returns true if it satisfies the condition. The nodes for which the predicate returns true are returned as a result of the current step. The predicate can start from the root node or from the relative path with respect to the current node. Putting (/) at the start of the predicate makes it start from the root node. A predicate can go up to any level in the XML value using the paths and return back to the original node after evaluating the condition. The following comparison expressions are supported in the predicate:

- ▶ Value comparison (ne, eq, gt, ge, lt, le)

These operators are used to compare the values of the same data types. For example:

A ne B

A and B should be of the same data type.

- ▶ General comparison (!=, =, >, <, <=, >=)

These operators are used to compare the sequence. If any of the sequence members satisfy the operator with the corresponding value in the other sequence, the operator returns true:

- (1,2)=(2,3) returns true.
- (1,2)=(3,4) returns false.

Similarly:

- (1,2)!(=)(2,3) returns true because at least one set is satisfying the operator.
- (1,1)!(=)(1,1) returns false.

Apart from these, you can use the logical operators AND and OR.

Constructors and FLWOR expressions

DB2 XQuery allows us to create our own XML structure from the existing XML values. To do so, it allows XML style constructors to create the XML document.

Example 2-12 shows a simple query that will get all of the five star movie titles (with rating as "*****") from the table and put into a new tag *movie-list*.

Example 2-12 Constructors in XQuery

```
xquery <movie-list rating='*****'>
{db2-fn:xmlcolumn('MOVIES.INFO')/movie/heading[rating='*****']/title} </movie-list>
```

Tip: When putting an XQuery inside the XML tag, enclose it in braces ({} to indicate that DB2 puts the result of the XQuery in the output; otherwise, DB2 takes the XQuery as a normal string and we see the normal XQuery text in the output.

DB2 supports FLWOR expressions to allow you more flexibility to restructure the existing XML values. FLWOR means *for*, *let*, *where*, *order by*, and *return*.

FLWOR expression syntax is:

```
FLWORExpr ::= (ForClause | LetClause)+ WhereClause? OrderByClause?
"return" ExprSingle
```

The *for* and *let* clauses in a FLWOR expression generate an ordered sequence of tuples of bound variables. While *for* iterates over the different values in the sequence, the *let* clause makes one binding with the value of full sequence.

The optional *where* clause serves to filter the tuple stream, retaining some tuples and discarding others.

Use the optional *order by* clause to reorder the tuple stream.

The *return* clause constructs the result of the FLWOR expression. The return clause is evaluated once for every tuple in the tuple stream, after filtering by the *where* clause and ordered by *order by* clause, using the variable bindings in the respective tuples.

Example 2-13 shows a simple FLWOR expression, which selects the five star (rating is "*****") movie's name.

Example 2-13 Using FLWOR expression

```
xquery for $i in db2-fn:xmlcolumn("MOVIES.INFO")/movie/heading where
$i/rating="*****" return $i/title @
```

You can nest the for and let clauses of the FLWOR expression to any level. The nesting of these clauses lets us combine different parts of the XML value.

Example 2-14 shows the nesting of these clauses in conjunction with the constructors. The query selects the movies based on their MPAA rating, MPAArating.

Example 2-14 Nesting FLOWOR expression

```
xquery for $ratings in
fn:distinct-values(db2-fn:xmlcolumn("MOVIES.INFO")/movie/movie-details/
MPAArating)
let $title:=db2-fn:xmlcolumn("MOVIES.INFO")/movie[movie-details/
MPAArating=$ratings]/heading/title
return <rating type='{ $ratings }'>
{ $title }
</rating>
```

Tip: When nesting the for and let clauses, keep in mind that the for clause creates one binding for each XML value in the sequence, and the let clause creates only one binding for the full sequence.

Use the order by clause to order the tuples based on the path expression. These ordered tuples are then processed by the return clause.

Example 2-15 gives an example of the order by clause. The query selects the movie name with the production studio order by the rating and the year of production.

Example 2-15 Using order by clause

```
xquery for $i in db2-fn:xmlcolumn("MOVIES.INFO")/movie order by
$i/movie-details/year
return
<movie>
{ $i/heading/title }
{ $i/movie-details/production/studio }
</movie>;
```

The FLWOR expression can be used to join XML values from different columns, too. Example 2-16 gives an example query, which selects the reviews from the MOVIEREVIEW table for all the movies from RR Pictures production.

Example 2-16 Joining two XML column values

```
Xquery for $id in
db2-fn:xmlcolumn('MOVIES.INFO')/movie[movie-details/production='RR
Pictures']/@id
let
$title:=db2-fn:xmlcolumn('MOVIES.INFO')/movie[@id=$id]/heading/title
return
  <movie title = '{$title}'>
    {
      for $review in
db2-fn:xmlcolumn('MOVIEREVIEW.REVIEW')/movie[@id=$id]/reviews
      return $review
    }
</movie>@
```

SQL/XML functions

The XQuery and SQL interfaces interact with each other using SQL/XML functions. Use these functions to embed the XQuery in the SQL/XML statement or SQL in the XQuery statement.

xmlquery function

You typically use this function in the column list of the SELECT statement to select a part of the document instead of the full value. This function is useful when you need to select the part of the XML value based on the condition on a relational column. Another significant advantage of this function is that it allows passing the relational column to the XQuery via its *passing by* clause. The table name for the column, to which the passing by clause refers, should be present in the FROM clause of the SELECT query.

Example 2-17 on page 74 gives an example for xmlquery function. The function selects the movie title with id=123.

Example 2-17 Using xmlquery function

```
select xmlquery('$d/movie/heading/title' passing movies.info as "d")
from movies where id=123
```

xmlquery function returns the single XML value for each row selected by the SELECT statement. The result of the xmlquery function should be a single XML

value per row of the result set (xmlquery function throws an error when the result is a sequence of XML values for a single row in the result set).

Tip: Use *xmlcolumn* and *sqlquery* resource functions inside the xmlquery function. But because these functions are applied to all the rows to concatenate the values first before applying the xquery path expression, the result of the xmlquery function might remain the same for all the rows selected by the SELECT clause.

To apply the XQuery on each individual row selected by the SELECT statement separately, use the passing by clause. The passing by clause passes the XML column value to the xquery for each row and returns the value along with other column values selected. All the tables to which the passing by clause refers should be there in the FROM clause of the SELECT statement.

xmltable function

Use this function generally in the FROM clause of the SELECT statement to create a table from the XML value. Similar to the xmlquery function, you can use the passing by clause here to pass any XML value or relational value to the function.

Example 2-18 shows an example that uses the xmltable function to create the relational table.

Example 2-18 Creating relational table using xmltable

```
select movie, writer, director from movies,
xmltable('$d/movie' passing movies.info as "d"
  columns
    movie varchar(20) PATH './heading/title',
    writer varchar(50) PATH './credits/writer',
    director varchar(50) PATH './credits/director'
  ) as T;
```

The xmltable function is useful when you want to move the data from an XML document to the relational table and you do not have the schema for the documents (or when you do not have annotated schema to do the same).

xmlexists predicate

xmlexists tests whether an XQuery expression returns a sequence of one or more items. xmlexists is similar to the xmlquery function except the return type. When xmlquery function returns the XML value, xmlexists returns the Boolean value. It returns true if the result of the xquery used in the function is not empty,

otherwise, it returns false. `xmlexists` also uses the `passing by` clause to pass the relational and XML column to the `xquery` and is useful when the filter condition is based on the comparison of the relational column with the XML node value. `xmlexists` is generally used in the `WHERE` clause of the `SELECT` statement.

Example 2-19 shows you a query, which returns the IDs for all the movies with genres Drama, Romance, and Remake.

Example 2-19 Using `xmlexists`

```
select id from movies where
xmlexists('$movie/movie/movie-details[genres="Drama, Romance and
Remake"]' passing movies.info as "movie")@
```

Use `xmlexists` in conjunction with the `xmlquery` function to select the part of the XML document based on a comparison between a node value and relation column value.

Example 2-20 shows you a query, which selects the movie name and the corresponding reviews as two separate columns of the result set.

Example 2-20 Using `xmlexists` in conjunction with `xmlquery`

```
select xmlquery('$d/movie/heading/title' passing movies.info as "d"),
xmlquery('$d/movie/reviews' passing moviereview.review as "d") from
moviereview, movies where
xmlexists('$d/movie[@id=$p]' passing moviereview.review as "d",
movies.id as "p")
```

xmlcast function

The `xmlcast` function is used to cast the XML value to other relational data types. It is equivalent to the existing `cast` function for relational data. `xmlcast` first resolves the XML value to an XQuery atomic data type and then does the conversion to the relational data type. Example 2-21 shows the usage of the `xmlcast` function. The resulting columns in Example 2-20 are of XML type. Using `xmlcast` function, the values can be casted to other data types.

Example 2-21 Using `xmlcast`

```
select xmlcast(xmlquery('$d/movie/heading/title' passing movies.info as
"d") as varchar(20)),
xmlcast(xmlquery('$d/movie/reviews' passing moviereview.review as "d")
as varchar(1000)) from moviereview, movies where
xmlexists('$d/movie[@id=$p]' passing moviereview.review as "d",
movies.id as "p");
```

xmlcast might throw an error if it is not possible to cast an XML value to the target data type.

2.2.9 Constructor function (publishing functions)

An XML value to be inserted into the XML type column can either be retrieved from the application or can be created from the other relational columns. DB2 provides publishing functions to create a new XML value from the existing relational columns.

Functions provided by DB2 to create a single node for an XML value include:

▶ **XMLELEMENT**

XMLELEMENT function can be used to create a new element node for an XML value. This function takes inputs, the name of the element, any attribute or namespace declaration, and the value of the node. The value of the node can be a complex type (concatenation of other elements) or a value of a relational column.

▶ **XMLATTRIBUTES**

XMLATTRIBUTES function is used to create the attributes of an element node. Because an attribute is always tied with the element, this function can be used only inside the XMLELEMENT function to create the attribute. This function takes input, the name of the attribute, and the value (any relational column).

▶ **XMLNAMESPACE**

XMLNAMESPACE is used to declare a namespace for an XML value.

▶ **XMLCOMMENT**

XMLCOMMENT function is used to create the comment node for the XML value. As a comment node can occur anywhere in the XML document, this function can be used anywhere to create the comment node.

▶ **XMLDOCUMENT**

XMLDOCUMENT function creates a document node for an XML value. Each XML value stored in the XML column must have a document node. A document node is associated with the XML value with a single root node. It is a logical way of making sure that the XML value has a single root node and not a sequence of XML nodes.

▶ **XMLPI**

XMLPI function creates a processing instruction node for the XML value.

► XMLTEXT

XMLTEXT function can be used to create a text node in an XML value. The input to this function is the value of the text node.

Functions to concatenate or aggregate more than one node include:

► XMLCONCAT

XMLCONCAT function concatenates the multiple XML nodes and returns the sequence of the nodes as a result. XMLCONCAT is a scalar function and only returns one XML value per row of the input expression. It takes only the existing XML value or column as input.

► XMLFOREST

XMLFOREST function creates a new list of element nodes. It takes the values of any data type as input and creates an element for each value given.

► XMLAGG

XMLAGG is an aggregate function. It takes a set of rows of XML values and produces a single XML result.

Example 2-22 shows how to use these functions to create an XML value from relational and XML columns. The example combines the information and reviews for a movie in a single XML value.

Example 2-22 Using publishing functions

```
select XMLELEMENT(NAME "moviereview",
    XMLCONCAT(movies.info,
    XMLELEMENT(NAME "reviews",XMLATTRIBUTES
        (moviereview.reviewid as "id"),
        XMLCONCAT(T.UserReview, T.CriticsReview)))
from movies, moviereview,xmltable('$d/movie/reviews' passing
moviereview.review as "d"
columns
    UserReview XML PATH './UserReview',
    CriticsReview XML PATH './CriticsReview') as T
where XMLEXISTS('$d/movie[@id=$id]' passing movies.id as
"id",moviereview.review as "d")
```

2.2.10 XML indexing

Indexes over XML data can improve the performance of queries on XML columns. Similar to relational index, an XML index over XML data indexes an entire column; however, there are some externally visible differences between relational indexes and XML indexes as follows:

- ▶ Indexes are created on columns of type XML based on path expressions (xmlpattern): a subset of XPath that does not contain predicates among other things.
- ▶ When creating an index, it is possible to specify what paths to index and what types. Use the type that you want to use in your queries.
- ▶ You can only index on a single XML column, composite indexes are not allowed at this time. Elements and attributes inside the document frequently used in predicates and cross-document joins can be indexed.
- ▶ If a node matches the xmlpattern but fails to cast to the specified index type, then no index entry is created for the node without raising an error.
- ▶ There could be zero, one, or multiple index entries per document (row) based on how many nodes match an xmlpattern.
- ▶ A single document can contain zero, one, or multiple nodes that match the xmlpattern. Thus, there can be zero, one, or multiple index entries for a single row in a table (significantly different than indexes on relational columns).
- ▶ Any nodes that match the path expression or the set of path expressions in XML that is stored in that column are indexed, and the index points to the node in storage that is linked to its parent and children for fast navigation.

XML values index

XML values index is a new type of index that can be created when users want efficient evaluation of xmlpattern expressions to improve performance during queries on XML documents. Unlike the traditional relational indexes where index keys are composed of one or more table columns specified by the user, the XML values index uses a particular XML pattern expression to index paths and values in XML documents stored in a single XML column.

You can also refer to an IBM white paper about XML indexing at the following Web site:

<http://www.ibm.com/developerworks/wikis/display/db2xml/Technical+Papers+and+Articles>

The following example shows the statement to create an XML index on XML data, which defines an index on all movie titles in all documents in the XML column *movieDetails* as shown:

```
CREATE INDEX title_idx on movies(movieDetails) generate key using  
xmlpattern '/movie/title' as sql varchar(50);
```

The *xmlpattern* is a path, which identifies the XML nodes to be indexed. It is called *xmlpattern* and not *xpath* because only a subset of the XPath language is allowed in index definitions. (Wildcards //, *, and namespaces are allowed, but XPath predicates such as /a/b[c=5] are not supported).

Since we do not require a single XML schema for all documents in an XML column, DB2 might not know which data type to use in the index for a given *xmlpattern*. Thus, you must specify the data type explicitly in the *as sql <type>* clause. The following types can be used:

► VARCHAR(*n*)

For nodes with values of a known maximum length.

► VARCHAR HASHED

For nodes with values of arbitrary length. In this case, the index contains hash values of the actual strings. Such an index can be used for equality predicates but not for range predicates.

The length of the data type VARCHAR specified when creating the index on an XML node value imposes a restriction on the length of the node value for XML data, which can be inserted into the table.

For example, if the index is created on a node with the data type as VARCHAR(5), inserting the XML data with the same node value with more than 5 characters throws an error.

Tip: Creating an index with the data type VARCHAR HASHED does not impose any restriction on the node value.

► DOUBLE

For nodes with any numeric type.

► DATE and TIMESTAMP

For nodes with corresponding XML values.

The following example defines a unique index on all movie ID attributes. Uniqueness is enforced within a document and across all documents in the XML column:

```
CREATE UNIQUE INDEX movie_id on movies(movieDetails) generate key using
xmlpattern '/movie/@id' atomic as sql double;
```

If an index definition contains the /text() in the XMLPATTERN as shown in the following example, then the query must also use the /text() in the predicate or the index does not match and cannot be used. Likewise, if the index definition does not contain the /text() in the XMLPATTERN, then the query should not use the /text() in the predicate. See the following example:

```
CREATE INDEX ratingidx on movies(movieDetails) generate key using
xmlpattern '/movie/movie-details/MPAARating/text()' atomic as sql
varchar(5)
```

XML full-text indexes

DB2 9 provides text search capability on both relational and XML data through Net Search Extender (NSE). NSE is enabled to support the new XML data type in DB2. You can follow the steps we outline to enable and use full text search with XML data:

1. The database itself needs to be enabled for full text search capability using the following command:

```
db2text enable database for text connect to dbname;
```

2. Create the full text index using the default or custom document model. A document model specifies a model for parsing and indexing structured documents of format HTML or XML:

```
db2text create index movietext for text on movies(movieDetails)
connect to dbname; (OR)
db2text create index movietext for text on movies(movieDetails)
format XML documentmodel XMLModel in file connect to dbname;
```

Specifying a model file allows you to limit indexing and search to subparts of the documents. Remember, full text search is not ready to use until the index is updated:

```
db2text update index movietext for text connect to dbname;
```

After creating and updating the text index the first time, the index needs to be kept up-to-date; that is, you must keep the content of the index in sync with the content of the base table. Indexes can be manually or automatically updated.

You can set an automatic update frequency for text indexes based on either or both the time of update and the minimum number of changes picked up from the log table (incremental changes are stored in the log table):

```
db2text ALTER INDEX movietext FOR TEXT UPDATE FREQUENCY d(1,2,3,4,5)
h(12,15) m(00) UPDATE MINIMUM 100
```

There are three types of full-text search supported by NSE:

- ▶ SQL scalar search function: SQL function for general text search application.
- ▶ SQL table-valued function: This supports general text search on presorted indexes and views.
- ▶ Text search stored procedure: This can be used for a high performance dedicated text search.

Among these three methods, we recommend SQL scalar function search for your search method and it is useful for most situations. You can integrate SQL search functions with DB2 optimizer for excellent performance where JOIN of data is needed. The basic syntax is:

```
SELECT column FROM table WHERE CONTAINS(column-name,
'search-criteria')=1
```

The following query exploits the index created in step 2 previously but restricts it to a specific element. The query retrieves all documents where the element /movie/movie-details/genres contains the word Comedy:

```
select movieDetails from movies where contains(movieDetails, 'sections
("/movie/movie-details/genres") "Comedy"') =1
```

The following example shows the use of full-text search function with xquery:

```
for $i in db2-fn:sqlquery('
  SELECT movieDetails FROM movies
  WHERE CONTAINS('movieDetails,
  SECTION("movie/title") "Dupree")=1')//movie/title
return $i
```

The following example shows the use of full-text search function with SQL or XML:

```
SELECT XMLQUERY('$t//movie/title' PASSING T.TITLE AS \"t\" )
FROM movies AS t
WHERE CONTAINS('movieDetails,
SECTION("movie/title") "Dupree")=1
```

To summarize, DB2 full-text search can help your application find relevant data based on structural and full-text search parameters. You can create and maintain

the text indexes easily either using command line tools or through DB2 control center.

2.2.11 Application support (interfaces)

The new XML data type requires existing application interfaces to recognize it. The major database programming interfaces, including Java and PHP, have been enabled and optimized to make use of DB2 XML data type. The XML is stored natively in pure hierarchical form; however, the data type has been externalized to the application interfaces as a serialized XML string. By default, all XML data accessed through the application interfaces is returned with an XML declaration, including an encoding attribute. The application can override this default code set if required. All of the major database interfaces support the XML data type as XML, not as a character type. This also avoids unwanted and unnecessary code page conversion. In this section, we briefly discuss various database programming interfaces supported by XML. This book is all about application development using various interfaces, so you can select a chapter of your interest for further reading.

JDBC

.JDBC or Java database connectivity layer has been enhanced to make XML data compatible with strings, byte arrays, and streams, so that columns and parameters can be bound to any of these types. Standardizing a JDBC XML type is a *work in progress*, but a proprietary XML type `com.ibm.db2.DB2Xml` is available right now. This interface provides a number of methods that make application development with XML easy. Look at the code in Example 2-23.

Example 2-23 JDBC with XML

```
String sql = "SELECT ID, MOVIEDETAILS from MOVIES where ID = ?";
PreparedStatement stmt = connection.prepareStatement(sql);
stmt.setString(1, "345");
ResultSet resultSet = stmt.executeQuery();
String xml = resultSet.getString("MOVIEDETAILS"); // or
InputStream inputStream = resultSet.getBinaryStream("MOVIEDETAILS");
// or
Reader reader = resultSet.getCharacterStream("MOVIEDETAILS"); // or
DB2Xml db2xml = (DB2Xml) resultSet.getObject("MOVIEDETAILS");
```

As you can see, the API provides various convenience methods to retrieve XML data, so that the application can process it as desired. Here is a list of DB2Xml methods for the JDBC API and a short description of their usage:

- ▶ `getDB2String()`:
Retrieves the value of the designated column in the current row of this `ResultSet` object as a string.
- ▶ `getDB2XmlString()`:
Retrieves the value of the designated column in the current row of this `ResultSet` object as a string with added XML declaration with encoding tag *ISO-10646-UCS-2*.
- ▶ `getDB2Bytes()`:
Retrieves the value of the designated column in the current row of this `ResultSet` object as UTF-8 encoded bytes.
- ▶ `getDB2XmlBytes(String targetEncoding)`:
Retrieves the value of the designated column in the current row of this `ResultSet` object as a byte array with the XML declaration with encoding tag.
- ▶ `getDB2AsciiStream()`:
Retrieves the value of the designated column in the current row of this `ResultSet` object as a stream of ASCII characters.
- ▶ `getDB2XmlAsciiStream()`:
Retrieves the value of the designated column in the current row of this `ResultSet` object as a stream of ASCII characters with encoding tag.
- ▶ `getDB2CharacterStream()`:
Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.io.Reader` object.
- ▶ `getDB2XmlCharacterStream()`:
Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.io.Reader` object with XML declaration with encoding tag *ISO-10646-UCS-2*.
- ▶ `getDB2BinaryStream()`:
Retrieves the value of the designated column in the current row of this `ResultSet` object as a UTF-8 encoded binary stream.
- ▶ `getDB2XmlBinaryStream(String targetEncoding)`:
Retrieves the value of the designated column in the current row of this `ResultSet` object as a binary stream. The driver converts the bytes to the `targetEncoding` and adds XML declaration with the encoding tag.

All XML APIs supported by Java language, including SAX (Simple API for XML), DOM (Document Object Model), and StAX (Streaming API for XML) are discussed in detail in the Java chapter. You can also refer to the DB2 information center for specific topics regarding application development with Java.

.NET data provider

The DB2 9 .NET interface supports the Microsoft .NET API and ADO.net data access APIs thoroughly. There is a chapter about application development using .NET in this book.

Call Level Interface

The DB2 Call Level Interface (CLI), a superset of ODBC, has been enhanced to support XML by providing a new SQL type, SQL_XML. Because there is no native XML type in C, the new SQL type can only be used in CLI/ODBC calls to mark XML values as XML type. In all other ways, access to serialized XML string data is identical to using a character array. The advantage is that the DB2 client and sever know that this is XML data and can avoid unnecessary or unwanted code page conversions. Example 2-24 shows how to retrieve and update XML data using the CLI interface.

Example 2-24 Retrieving and updating XML data using CLI

```
char buf[10240];
integer length;
// retrieve XML
SQLExecute( hStmt, "Select movieDetails from movies where id = '345'",
SQL_NTS );
SQLBindCol( hStmt, 1, SQL_C_BINARY, buf, &length );
SQLFetch( hStmt);
SQLClose( hStmt);

// update as XML
SQLPrepare(hStmt, "update movies set movieDetails = ? where id =
'345'", SQL_NTS );
SQLBindParameter( hStmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
SQL_XML, buf, &length);
SQLExecute( hStmt );
```

Please note that data is fetched and inserted as SQL_C_BINARY.

Embedded SQL interface

The SQL standard defines new host variable declarations for XML types, and DB2 is using this in its implementation. Example 2-25 on page 86 shows the embedded SQL with XML.

Example 2-25 Using XML in embedded SQL

```
EXEC SQL BEGIN DECLARE;
    SQL TYPE IS XML AS CLOB( 10K ) xmlBuf;
    SQL TYPE IS CLOB( 10K ) clobBuf;
EXEC SQL END DECLARE SECTION;

// as XML
EXEC SQL SELECT movieDetails INTO :xmlBuf from movies where id = '345';
EXEC SQL UPDATE movies SET movieDetails = :xmlBuf where id = '345';

// as CLOB
EXEC SQL SELECT XMLSerialize( movieDetails ) INTO :clobBuf from movies
where id = '345';
EXEC SQL UPDATE movies SET movieDetails = XMLParse(:clobBuf) where id =
'345';
```

2.2.12 Utilities and XML support

In DB2 9, DB2 utilities and commands are modified for XML support.

XML import and export

IMPORT and *EXPORT* commands are updated for XML options. The *IMPORT* and *EXPORT* command treats the XML value similar to the LOBs. Equivalent to the LOB's options *LOBFILE*, *LOBS TO* and *LOBS FROM*, *XMLFILE*, *XML TO* and *XML FROM* options are added for the XML values. The only difference between the LOBs and XML export is that unlike LOBS, an inline XML value in the exported data file is not possible. So the XML values are exported to the separate files always. Similar to LOB location specifier (LLS) for LOB data, an XML data specifier (XDS) is introduced to define the XML value in data files. The attributes of the XDS are *FIL*, *OFF*, *LEN*, and *SCH*, which we explain as follows:

▶ **FIL**

This attribute gives the file name where the XML values are exported.

▶ **OFF**

This attribute specifies the offset of the XML value in the file in case XML values are concatenated in a single file.

▶ **LEN**

This attribute specifies the length of the XML value.

▶ **SCH**

For *IMPORT*, this attribute gives the XML schema relational name, which is used to validate the XML schema while importing the data to the table. To

export this attribute, give the schema relational name, which was used to validate this XML value when it was inserted.

IMPORT syntax

We explain the new options for the **IMPORT** command using the following command:

```
IMPORT FROM export.del OF DEL XML FROM xmlpath MODIFIED BY XMLCHAR  
XMLPARSE PRESERVE WHITESPACE XMLVALIDATE USING XDS DEFAULT SCHEMA_A,  
IGNORE (SCHEMA_B, SCHEMA_C, SCHEMA_D) MAP ((SCHEMA_E, SCHEMA_F),  
(SCHEMA_G, SCHEMA_H)) INSERT INTO T1
```

The explanations for the terms in the command are:

- ▶ **XML FROM** option
Specify one or more paths where the XML files are stored.
- ▶ **MODIFIED BY XMLCHAR** option
Specify that the XML data in the files is in application character code page.
- ▶ **MODIFIED BY XMLGRAPHIC** option
Specify that the XML data in the files are in application graphic code page.
- ▶ **XMLPARSE** option
Indicate that the whitespace in the XML values would be removed or not. **STRIP WHITESPACE** removes the whitespaces and the **PRESERVE WHITESPACE** option preserves the whitespaces in the XML value.
- ▶ **XMLVALIDATE** option
Specify how to validate the XML values before importing them in the table column. There are three options to validate an XML value:
 - **XMLVALIDATE USING XDS** option
Specify that the XML value is validated using XDS's SCH attribute. Use the **DEFAULT** clause of this option to specify the default schema name in case XDS does not contain the SCH attribute. Use the **IGNORE** clause to ignore some schemas. In case the SCH attribute value is any, the schema mentioned in the **IGNORE** clause, validation is ignored and the value is imported without any validation.

Use the **MAP** clause to map a schema to other schema. The **MAP** clause has a set of schemas for each entry. If the SCH attribute value is any of the schema mentioned in the first schema name in the set of schema values in the **MAP** clause, the schema is mapped to the second schema entry in the same set and the validation is done against the mapped schema.

Tip: For a particular row, you can only apply one of the DEFAULT, IGNORE, or MAP clauses. The priority of these clauses is DEFAULT, IGNORE, and MAP. This means that if the XDS does not contain the SCH attribute, the schema in the DEFAULT clause is used and cannot be further mapped or ignored. Similarly, a schema, which is already mapped, cannot be ignored. It might be possible that the SCH attribute for rows has the same value as the DEFAULT clause. In that case, the schema can be ignored or mapped because the DEFAULT clause is not applicable to that.

- XMLVALIDATE USING SCHEMA option
Specify that the XML value should be validated against the schema specified here. The SCH attribute is ignored in every case.
- XMLVALIDATE USING SCHEMALOCATION HINTS option
Specify that XML values are validated using the schemalocation hints present in the XML value itself. The SCH attribute is ignored.

EXPORT syntax

We explain the new option for the **EXPORT** command using the following command:

```
EXPORT TO export.del OF DEL XML TO /xmlpath XMLFILE xmldocs MODIFIED BY  
XMLINSEPFILES XMLCHAR XMLSAVESCHEMA SELECT * FROM T1
```

The explanations of the terms in the command are:

- ▶ XML TO clause
This clause provides the paths where the XML data is stored. The default value for this clause is the path where the data file is written. Multiple values can be given for this clause. If the multiple values are provided, export will cycle between the paths to write each column value to the appropriate XML file.
- ▶ XMLFILE option
This option supplies the base name of the XML file where the XML values are written. The default value for the base name is the name of the data file. The full name of the file consists of the base name followed by a number, which is padded to three digits and then the three letter identifier (XML). For the previous command, the file name is xmldocs.001.xml.

- ▶ **MODIFIED BY** clause
Use this to change the behavior of exported files, and this is called a file type modifier. We introduce the following new file type modifiers.
- ▶ **MODIFIED BY XMLINSEPFFILES** option
This option specifies that the XML value for each column should be stored on the separate files instead of storing it in a single file. By default, all the values are concatenated in a single file. If you specify this option, a separate file name is generated for each column to store the XML value. For the previous command, the file names are `xmldocs.001.xml`, `xmldocs.002.xml`, and so on.
- ▶ **MODIFIED BY XMLGRAPHIC** option
This option specifies that the XML data should be written in the application graphics code page.
- ▶ **MODIFIED XMLCHAR**
This option specifies that the XML data should be written in the application character code page.
- ▶ **MODIFIED BY XMLNODEDECLARATION**
This option specifies that no XML declaration tag should be added to the XML value. By default, every XML value written to the file has a declaration tag with encoding information.
- ▶ **XMLSAVESHEMA**
This option specifies that the schema name should be saved for all XML columns. The fully qualified name of the schema is stored as XDS's SCH attribute.

XML RUNSTATS

The **RUNSTATS** command supports XML value and collects information regarding the XML data in the tables and indexes created on an XML value. The **RUNSTATS** command for an XML value runs in the same way it runs for the relational columns. The XML column supports the following **RUNSTATS** commands:

```
RUNSTATS on table <schemaname.tablename>
RUNSTATS on table <schemaname.tablename> on columns <column list>
RUNSTATS on table <schemaname.tablename> on columns <column list> with
distribution on columns <column list>
```

RUNSTATS collects the information regarding the XML columns if either **RUNSTATS** is run on the full table or the name of the XML column appears in *<column list>*.

2.2.13 XML type support in stored procedures

Stored procedures are enabled to accept the XML parameter values.

SQL stored procedures

In SQL, stored procedure parameters of type XML look just like the variables of other data type.

Example 2-26 shows a simple SQL stored procedure, which takes three input parameters. The first parameter is of XML type and the other two are of integer type. The procedure checks if the ID value in the XML parameter `review` is equal to the ID values passed as the argument `ID`. If yes, the procedure inserts the review in the `MOVIEREVIEW` table with the review ID equal to the value of the `reviewid` parameter.

Example 2-26 Stored procedure with XML data type input

```
create procedure proc1 (IN review XML, IN id int, IN reviewid int)
language SQL
BEGIN
DECLARE var1 XML;
if(XMLEXISTS('$x/movie[@id=$d]' passing review as "x", id as "d"))
then
insert into moviereview values(reviewid, review);
end if;
end
```

Example 2-27 calls the procedure that was created in Example 2-26.

Example 2-27 Call stored procedure passing XML parameter

```
call proc1(XMLPARSE(DOCUMENT('<movie id="111">
<reviews>
<UserReview>
<user name="Andy">Three is a crowd, yet there are not enough stars to support
this catastrophe.</user>
<user name="Linda">Three is a crowd, yet there are not enough stars to support
this catastrophe.</user>
</UserReview>
<CriticsReview>
<newspaper name="ABC Times">The best mystery thriller of the year. </newspaper>
<newspaper name="San Francisco Post"> The best mystery thriller of the year.
</newspaper>
</CriticsReview>
</reviews></movie>')),111,222)@
```

Because XQuery does not support static cursors, in SQL stored procedures, it is not possible to declare the cursor for the XQuery statement using the DECLARE CURSOR statement, which means the following command gives an error.

```
DECLARE cur1 CURSOR for XQUERY .....
```

To create a cursor for an XQuery statement in an SQL stored procedure, do this:

```
DECLARE stmt_text VARCHAR (1024);  
DECLARE stmt STATEMENT;  
DECLARE cur1 CURSOR FOR stmt;  
SET stmt_text = <xquery statement>;  
PREPARE stmt FROM stmt_text;
```

XML type in external routine

You can use the XML data type in externally stored procedures and user defined functions. You declare the parameters in the same manner that you declare the host variable for the programming language used for the external routine. For details about how to declare the host variables in different programming languages, refer to 2.2.11, “Application support (interfaces)” on page 83.

Example 2-28 gives the command, which registers a stored procedure written in C language.

Example 2-28 Registering a stored procedure

```
CREATE PROCEDURE proc2(IN review XML AS CLOB(2M), IN id int, IN  
reviewid int)  
LANGUAGE C  
FENCED  
PARAMETER STYLE SQL  
PARAMETER CCSID UNICODE  
EXTERNAL NAME 'proclib!proc1';
```

In external routines, you should serialize XML values before sending the value to the user’s code. To avoid the code page conversion, XML values should be passed in unicode format. To specify that, routines with the XML type parameter should be declared as PARAMETER CCSID UNICODE.

C routines only support the XML AS CLOB type parameter.

For more details about how to handle XML variables, refer to the respective chapter.



Application development with PHP

The PHP language has been a darling of the open source software development community for years for reasons such as performance, the power of the language itself, and ease of use. Now, PHP language with the frameworks being built around it, seems to be ready for enterprise software development. This chapter provides you concept and hands-on with Zend Framework for building enterprise Web applications that require database and application integration support. When deciding to build an On Demand¹ Web application, there are many products and technologies that you can use. In most of the cases, you need to choose the following:

- ▶ Web server software
- ▶ An operating system
- ▶ A database management system
- ▶ A programming or scripting language

In this chapter, we do not address the hardware, operating system, or Web server software much. In fact, we do not need to. One of the strengths of PHP is that it is available on all of today's popular operating systems, including many versions of UNIX and Linux operating systems. Similarly, IBM's flagship

¹ An On Demand Business is an enterprise whose business processes, integrated end-to-end across the company and with key partners, suppliers and customers, can respond with flexibility and speed to customer demand, market opportunity, or external threat.

database product, DB2, is also versatile and runs on Microsoft Windows, UNIX, Solaris, and many flavors of Linux. We have used the two most popular and widely used setups to showcase the application development steps in this chapter:

- ▶ SUSE Linux Professional with Apache Web Server
- ▶ Microsoft Windows XP Professional using Apache Web Server

Irrespective of the Web server software and operating system you choose, you should seriously consider using PHP with DB2 Express-C for the following reasons:

- ▶ No charge

Both PHP and DB2 Express-C are free. You can download the latest versions of DB2 Express-C and PHP from the following Web sites absolutely free of charge:

- <http://www.ibm.com/db2/express>
- <http://www.php.net>

Alternatively, you can use Zend Core for IBM, which packages the Apache Web Server, PHP libraries, and DB2 in one bundle. Currently, Zend Core include DB2 V8.2, so we recommend that you install DB2 9 first and then try to install Zend Core for IBM. Zend Core is supposed to find the DB2 already installed on your system and use it unless you choose to replace it.

Remember that in order to take advantage of DB2 pureXML technologies and the XCS (discussed in this chapter), you must use DB2 9. We expect that Zend Core for IBM will be available with DB2 9 by the time this book is published. Currently, Zend Core for IBM is available for free download at the following Web site:

http://www.zend.com/products/zend_core/zend_core_for_ibm

- ▶ Ease of development

DB2 Express-C supports a full range of APIs for application development and simplifies the server side development with stored procedures and SQL functions. For PHP developers, DB2 Express-C comes bundled with Zend Core for IBM, a seamless, ready to use, easy-to-use PHP development environment.

- ▶ Support for XML

DB2 Express-C includes leading edge technology for storing, managing, and searching XML data in a secure scalable environment. With its pureXML technology, your PHP application makes use of standard XQUERY and SQL/XML to build the latest applications.

- ▶ Performance

Both PHP and DB2 Express-C are very fast and efficient. PHP Web pages can serve millions of hits per day using a single inexpensive server. DB2 Express-C is proven to be faster than its competition in most of the application development scenarios.

- ▶ Portability

Both PHP and DB2 Express-C are available on many platforms. Your code usually works without modifications on different systems that run PHP and DB2.

- ▶ Information on demand

Information on demand is about getting the right information to the right people at the right time to take advantage of the business opportunities. Both PHP and DB2 Express-C are extremely flexible in supporting the varying demand of business requirements. Also since Express C has been enabled to tightly integrate with other IBM Express products, you can leverage this to build scalable and resilient on demand solutions very quickly.

3.1 Application environment

In the true spirit of the open source movement, we decided to build our sample application using Web Development Framework.

In software development, a framework is defined as a support structure in which other software projects can be organized and developed. A framework might include support programs, code libraries, a scripting language, or other software to develop and glue together the different components of a software project.

Here are few examples of application development framework. You can find more than 20 frameworks on the Web, written in PHP alone:

- ▶ *Apache Cocoon* and *Apache Struts* from Apache software foundation
- ▶ *CakePHP*, an open source framework written in PHP
- ▶ *Zend Framework*, an open source framework in PHP (preview release)
- ▶ *Eclipse* framework from Eclipse foundation
- ▶ *.Net* framework from Microsoft
- ▶ *Ruby on Rails*, an open source framework written in Ruby
- ▶ *Net Beans*, a Java-based Web application development framework by Sun Microsystems™, Inc.
- ▶ *Catalyst*, an open source development framework written in Perl
- ▶ *Django*, an open source Web application framework written in Python
- ▶ *Cocoa* from Apple computer

Why framework?

A framework provides a skeleton on which you can build a software solution using different components. A framework also allows developers to focus on business logic and not the tedious part of software integration. Some people may argue that using a framework can add code *bloat* (unnecessary long and slow code) and also usually involves a steep learning curve. But the increasing popularity of PHP for database-driven Web applications is enough good reason for us to consider use of a framework for PHP development in this book. We decided to use an open source framework called *Zend Framework for PHP*. We divide this chapter into two parts.

Part I gives an overview of PHP application development using Zend Framework preview release, followed by a sample database application using Zend DB Adapter for access to DB2 Express-C.

Part II focuses on development of a PHP application using the new XCS (XML Content Store) technology, which is part of Zend Framework. This section is also followed by a sample, Social Network Application, which you can build and deploy very quickly using XCS and new DB2 Express-C XML pureXML technology.

Again, in the spirit of open source, we decided to use the Eclipse application development environment (with a PHP plug-in) to write PHP code. Zend also offers a trial version of its development environment called Zend Studio, but we chose Eclipse because it is free and easy to use.

3.1.1 Zend Framework overview

The PHP language has been around for a long time, but it was not adopted in large enterprise environments. It began as a scripting language and was considered as a glue to integrate processes quickly and flexibly. PHP is also popular for building database driven Web applications. With the development of application development frameworks, PHP is well prepared for enterprise applications because it should be capable of integrating components and services in the enterprise. Zend Framework is this type of framework developed by Zend and its partners for designing PHP5-based enterprise grade Web applications quickly. The framework is based on MVC (Model-View-Controller) architecture. The primary benefits of using a MVC architecture for your next Web application are:

- ▶ The same enterprise data can be accessed using different views. It accesses enterprise data through the model and specifies how that data should be presented. It is the view's responsibility to maintain consistency in its presentation when the model changes.
- ▶ The same enterprise data can be updated using different views.
- ▶ Core business logic is separated from the presentation layer and control logic. The controller translates interactions with the view into actions for the model to perform.

Zend Framework aims to provide an architecture for developing entire Web applications with no other library dependencies. Zend and its partner companies are committed to actively develop and support the framework code. Zend also supports a community Wiki page as a developer zone. This Web site contains recent committed code and also issues problem (bug) tracking and other developer resources. Zend Framework also enforces a strict PHP coding standard to maintain uniformity among the framework and application code. Refer to the complete PHP coding standard at the Zend Web site:

<http://framework.zend.com/manual/en/coding-standard.html>

The core of Zend Framework is the *Zend_Controller*. The *Zend_Controller* is designed to be lightweight, modular, and extensible. The *Zend_Controller* workflow is implemented by several components. Understanding the underpinnings of all components is not necessary to build an application, but knowing how they work and interact with each other is definitely helpful:

▶ *Zend_Controller_Front*

This component processes all the requests received by the server and is responsible for delegating requests to action controllers.

▶ *Zend_Controller_Router*

This process takes the URI endpoint and decomposes it to determine which controller and action of that controller should receive the request.

▶ *Zend_Controller_Dispatcher*

The dispatcher process takes the dispatcher token, finds the appropriate controller, instantiates the controller class, and finally runs the action method in that controller object.

In summary, this is how *Zend_Controller* works:

1. A request is received by *Zend_Controller_Front*, which in turn calls *Zend_Controller_Router* to determine which controller (and action in that controller) to dispatch.
2. *Zend_Controller_Router* decomposes the URI into a *Zend_Controller_Dispatcher_Token* object that describes where to dispatch.
3. *Zend_Controller_Front* then enters a dispatch loop. It calls *Zend_Controller_Dispatcher*, passing it the token, to dispatch to the actual controller and action.
4. After the controller has finished, control returns to *Zend_Controller_Front*.
If the controller has indicated that another controller should be dispatched by returning a new token, the loop continues and another dispatch is performed. Otherwise, the process ends.

We discuss more about these objects whenever needed, but for now, let us go through the setup required for using Zend Framework.

3.1.2 Setting up Zend Framework

Setting up Zend Framework is quite easy. This section assumes that you have PHP up and running with some supported Web server in your environment. Our example uses:

- ▶ Apache HTTP server 2.0.58 and assumes it is installed in the C:\Program Files\Apache Group\Apache2 directory.
- ▶ PHP 5.1.4 with the latest IBM DB2 driver for PHP.

If you need help with setting up Apache HTTP server with PHP, we provide the setup procedures in Appendix A, “Setup procedure and sample data” on page 295.

You need to download and save the preview version of Zend Framework from the following Web site:

<http://framework.zend.com/download>

Note: Zend Framework is currently released as preview 0.1.4 and at the time of writing this book, this was the release available to use. Check the Web site and use the latest version as it becomes available. You can also connect to their Subversion Repository for anonymous checkout of the latest code and report bugs. Zend Framework Developer Web site has plenty of information for PHP developer.

In order to get started with our first Web application, let us follow these steps carefully:

- ▶ Directory structure

One very important goal of any framework is to simplify the common code structure. We recommend this directory structure:

- a. Save and unzip the Zend Framework preview code in the directory of your choice. When unzipped, it saves itself in ZendFramework-0.1.4 directory and contains demos, documentation, incubator, library, and tests directories along with a readme, news, and license files.
- b. Create a directory called zframework in the Apache Web server htdocs directory as follows:

```
C:\Program Files\Apache Group\Apache2\htdocs\zframework
```
- c. Copy the entire library directory from your Zend Framework directory to the zframework directory you just created. Rename this directory to lib.

- d. Create additional directories in zframework directory to organize your code. The directory structure should look as shown in Example 3-1.

Example 3-1 zframework directory structure

```
/app
  /models
  /views
  /controllers
/www
  /images
  /styles
  .htaccess
  index.php
/lib
/Zend
```

► Web server configuration

One important goal of modern Web applications (Web sites) is to have clean URIs (without any attached query parameters). This kind of URI rewrite support is required from Web server. Zend Framework currently depends on Apache's `mod_rewrite` module to redirect all requests to a single file called `index.php` that resides in Web server's document root directory and contains the bootstrap function (in this case, `Zend_Controller_Front`). The configuration of the server requires a few steps:

- a. The first step is to enable `mod_rewrite` function of Apache Web server. By default, this function is not enabled. To enable it:
 - Uncomment `LoadModule` for `mod_rewrite` function in the Apache configuration file `httpd.conf`.
 - You would also need to set the Document Root to `.../zframework/www` and change `AllowOverride` value from `NONE` to `ALL`.

After these changes, restarting Web server should enable the `mod_rewrite`.

Note: Zend is currently looking for a solution, which does not require `mod_rewrite`, for the future. There is a good chance that the final release of this framework will not require you to perform this setup.

- b. Next, we need to set the rule to redirect all incoming Web requests to `www/index.php`. You can easily do this by creating a `.htaccess` file in the `www` directory with the contents as shown in Example 3-2.

Example 3-2 .htaccess file

```
RewriteEngine on
RewriteRule !\.(js|ico|gif|jpg|png|css)$ index.php
```

The above file redirects all requests not containing one of the previous file extensions to `index.php` bootstrap file.

Alternatively, you can modify `httpd.conf` also for the rewrite rules, but that requires you to restart the Web server. Adding to `.htaccess` file does not require you to restart Web server.

Note: This is the only php file required to exist within the Web server document root directory. Also for security reasons, it makes sense *not* to store the php files in the directories that are accessible by the Web server.

You also need to include the `include_path` to the framework library directory (`zframework/lib`). You can add this to the `php.ini` file or simply add it into `.htaccess` as shown in Example 3-3.

Example 3-3 include_path

```
php_value include_path "C:\Program Files\Apache
Group\Apache2\htdocs\zframework\lib"
```

► Zend class

The Zend class (`zend.php`) contains static methods that are used by many classes. This is the only class you need to include manually, so we add it to our bootstrap file `index.php`.

After adding it to the bootstrap file, we now have access to all static methods. You can load other classes with the `loadClass()` method. For example, to load `Zend_Controller_Front`:

```
Zend::loadClass('Zend_Controller_Front');
```

Now, to understand the framework flow and make sure setup is working, let us do this exercise:

- a. Edit your index.php to look as shown in Example 3-4.

Example 3-4 index.php

```
<?php
include 'Zend.php';
function __autoload($class)
{
    Zend::loadClass($class);
}
$dbuser = "db2admin";
$dbpass = "db2admin";
$dbname = "CONTACTS";
$params = array( 'username' => $dbuser,
                'password' => $dbpass,
                'dbname' => $dbname );
$conn = Zend_Db::factory('Db2', $params);
$db = new Database($conn);
Zend::register('db', $db);
$view = new Zend_View;
$view → setScriptPath('../app/views');
Zend::register('view', $view);
$controller = Zend_Controller_Front::getInstance()
             → setControllerDirectory('../app/controllers')
             → dispatch();
?>
```

Let us discuss the different parts here:

- i. First we include the zend.php.
- ii. Next, we are using PHP5's autoloader function to load the necessary controller classes.

In PHP5 and above, you can define an autoloader function (`__autoload`) which is automatically called in case you are trying to use a class which has not been defined yet. The scripting engine is given a last chance to load the class with the `__autoload` function before PHP fails with an error.

- iii. Next we create and instantiate a database connection. `Zend_Db_Adapter` provides the database API abstraction layer, which supports many SQL databases including DB2. You need to call `Zend_Db::factory()` with the adapter name (DB2 in this case) and a list of parameters with connection information. We now save this db adapter `$db` in the Zend registry variable so that we can access it in our application anywhere.
 - iv. Next we create an instance of `Zend_View` and set the path to the instance to tell where to look for view template files. `Zend_View` is a class for working with the "view" portion of the model-view-controller pattern. That is, it exists to help keep the view script separate from the model and controller scripts.
 - v. The last section of `index.php` creates the instance of the front controller and specifies the directory where actual controller files are found. The framework wants you to create controller classes in `app/controllers` directory.
- b. Now let us try to point your Web browser to the Web server with some random path such as `http://localhost:81/redbook/toc`. In our case, you should see an error. We expect the error and here is why. Since we requested for `redbook/toc`, the framework looks for a controller called `RedbookController.php` in the `app/controllers` directory. Since it did not find any, it then looked for default `IndexController.php`. It did not find it either. Remember, every request is treated as controller/action in the framework. `Index` is the default for both the controller and the action. So now, let us create `IndexController.php` as shown in Example 3-5 to get going.

Example 3-5 IndexController.php

```
<?php
Zend::loadClass('Zend_Controller_Action');
class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
        echo 'what should I do?';
    }
}
?>
```

There is another useful method called `noRouteAction()`. You can specify a `_redirect()` call to redirect the requests for nonexistent pages to the desired page. For example, as shown in Example 3-6 on page 104, if you

add the `noRouteAction()` method to the `IndexController.php`, it redirects our request of `redbook/toc` to the main page.

Example 3-6 Redirect request

```
<?php
Zend::loadClass('Zend_Controller_Action');
class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
        echo 'what should I do?';
    }
    public function noRouteAction()
    {
        $this->_redirect('/');
    }
}
?>
```

- c. Now, let us create the `RedbookController.php` as shown in Example 3-7. We have an `indexAction()` method for non-existent action request and `tocAction()` for our `redbook/toc` request.

Example 3-7 RedbookController.php

```
<?php
Zend::loadClass('Zend_Controller_Action');
class FooController extends Zend_Controller_Action
{
    public function indexAction()
    {
        echo 'What should I do?';
    }
    public function tocAction()
    {
        echo 'TOC of this book follows';
    }
    public function __call($action, $arguments)
    {
        echo 'RedbookController: __call()';
    }
}
?>
```

Now, our request for `http://localhost/redbook/toc` should echo the text `T0C of this book follows`. Notice there is another function, `__call()`, to handle undefined actions such as `redbook/appendix`.

You can see that the controller/action mechanism allows you to have friendly URIs for your Web application and also allows you to organize them properly. This completes our basic test of `mod_rewrite` and the framework setup. We discuss other controller objects in detail when we build our sample application but for now we move on with the Eclipse environment setup.

3.2 DB2 Interface with PHP

There are three main extensions of PHP that you can use to develop applications with DB2:

- ▶ `ibm_db2`
- ▶ `PDO_ODBC`
- ▶ Unified ODBC

IBM recommends using `ibm_db2` or `PDO_ODBC` to get best results out of DB2 database.

IBM_DB2

`ibm_db2` extension of PHP provides an interface to connect from PHP to IBM DB2 database. `ibm_db2` provides a mechanism to connect to both cataloged and non-cataloged databases. This extension provides mechanisms for application developers to issue SQL queries, work with large objects, call stored procedures, use persistent connections, and use prepared SQL statements. It also works on PHP releases below Version 5. Unlike `PDO_ODBC`, `ibm_db2` is based on traditional procedural programming and performs better compared to Unified ODBC functions. `ibm_db2` provides built-in functions for getting details about the DB2 database server and client by querying system catalog tables, which provide lots of information about the DB2 database management system. The latest `ibm_db2` extension for PHP has been enhanced to support the new XML data type in DB2 V9. We highly recommend that you download the latest `ibm_db2` extension from the following Web site:

http://www.pec1.php.net/package/ibm_db2

You can download the library(dll) for Windows from the following Web site:

http://pec14win.php.net/ext.php/php_ibm_db2.dll

PDO_ODBC

PDO (PHP Data Objects) is an object-oriented, standards-based data access method in PHP, where you can use the same methodology to query the database and fetch data from the supported databases. PDO_ODBC extension is the implementation of the PDO specification. When compiled with DB2 libraries, you can use it to access DB2, Cloudscape™, and Apache Derby databases. This extension provides a mechanism to connect to both local cataloged and non-local (remote) cataloged databases. For a local cataloged database, PDO_ODBC obtains the database server details from the client machine. For a non-local cataloged database, the full details of the remote database are specified in the connection URL. PDO_ODBC also provides access to advanced features of DB2, such as persistent connections, prepared SQL statements, large objects, and stored procedures. It provides better performance compared to Unified ODBC functions.

Unified ODBC

Unified ODBC was the only method for PHP to talk with DB2, Cloudscape, and Apache Derby databases before `ibm_db2` or PDO_ODBC was released. Like `ibm_db2` and PDO_ODBC, this extension also interacts with DB2 using native CLI calls. It uses the same PHP methods to interact with different databases even if the underlying mechanism is different. But you cannot use this API to call a stored procedure in DB2. Unified ODBC does not use the Object-Oriented methodology.

The source code for all the extensions is available for free download in the PECL Web site:

<http://pecl.php.net/>

3.3 Setting up Eclipse with PHP

Eclipse is an open source application development tool that you can download at no charge from the following Web site:

<http://www.eclipse.org>

This tool is ready to use once you download and unzip it in a folder of your choice. There is no setup required. We advise you to make a desktop shortcut of `eclipse.exe` to make the launch easy. After launching Eclipse from the desktop, it takes you to the Welcome window where you can navigate the built-in help and samples. The core Eclipse tool does not come with a PHP plug-in.

You need to configure it as follows:

1. From the Eclipse menu, select **Help** → **Software Updates** → **Find & Install**.
2. Select **Search for New Feature to install**, then click **Next**.
3. Click **New Remote Site**.
4. In the window that appears, enter the following information and click **OK**:
 - a. In the Name field, type PHP Plug-in
 - b. In the URL field, type:
`http://phpeclipse.sourceforge.net/update/releases`
5. Make sure the PHP Plug-in is checked, and then click **Finish**.
6. On the next window, accept the license agreement and click **Next**.
7. Click **Finish**.

The Eclipse Update Manager downloads and prompts you to select the component and confirm installation. Once installed, you need to restart Eclipse to be able to use PHP features.

The Eclipse window in Figure 3-1 on page 108 shows the tool with the PHP menu and functions. You can also switch the Eclipse window perspective to switch to another development environment, such as Java. For more information about using the Eclipse tool, see the Eclipse documentation. The major advantage of using Eclipse with the PHP plug-in is that it provides better organization of your code with color-coded text for errors and warnings. It also includes the PHP Debug environment to debug the code and a PHP browser to view the result.

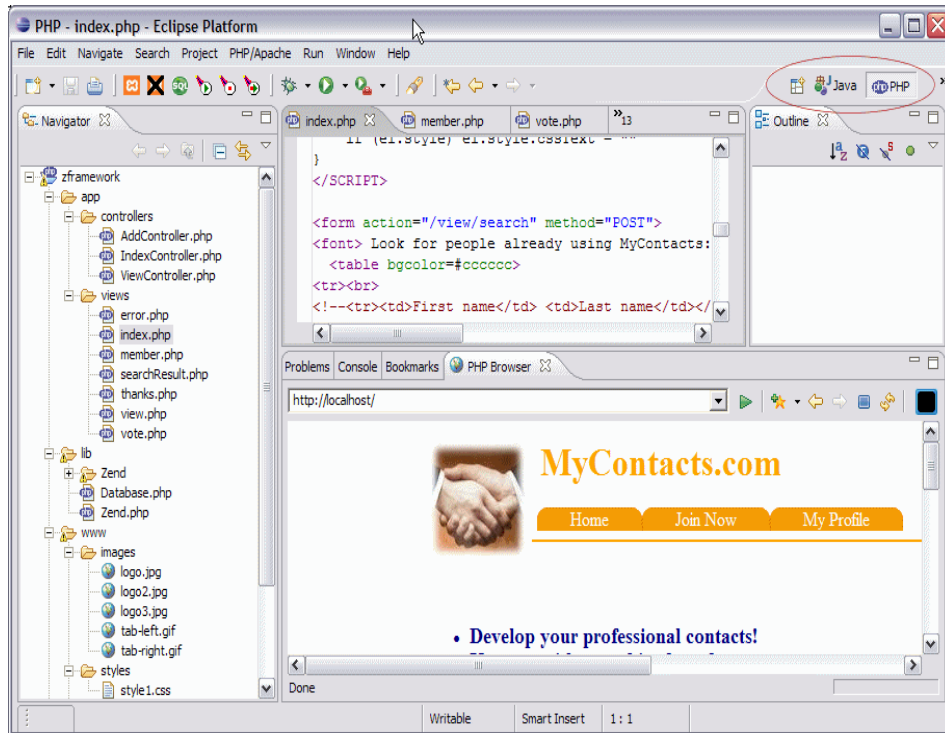


Figure 3-1 Eclipse

3.4 Sample Web application

In order to showcase the features of DB2 Express-C, PHP, and the brand new Zend Framework, we built a couple of sample Web-based database applications. The first application is a “Movie of the week” survey application, where using a simple Web interface, you can add a movie name to the database and vote for the movie of the week. Your suggestion for a new name or your vote instantly updates the Web site. The application demonstrates the framework’s simplistic MVC model to build database driven applications using Zend db adapters.

While designing any Web-based application, we start by designing interfaces. A Web application is a collection of Web pages where each page is a unique URL. This simple survey application consists of two URLs:

- ▶ /index/vote
- ▶ /add/movie

You need to think of these URLs in terms of controller/action. The `IndexController` fetches the existing movies' names and the respective votes from the database and `AddController` handles adding new suggestions and updating votes. You can have multiple presentation layers created in the `views` directory, but for simplicity, we have a single presentation layer `vote.php` which manipulates and presents the survey results.

Table 3-1 lists the files created for our movie example. You can download the complete zip file from the redbook Web site. For download details, see Appendix C, "Additional material" on page 319.

Table 3-1 Files for example application

Name	Type	Description
<code>IndexController.php</code>	Controller	Main controller
<code>AddController.php</code>	Controller	Contains add/update logic
<code>vote.php</code>	View	Presentation HTML

Let us look at the `IndexController.php` (Example 3-8) first. We have intentionally left `indexAction` method untouched and created a `voteAction()` method. As a result of this, the main URL to launch this application is:

`http://localhost/index/vote`

You can alternatively not have the `voteAction()` method and move the code inside the `indexAction` method(). In that case, you launch the application with this URL:

`http://localhost/`

Example 3-8 `IndexController.php`

```
<?php
Zend::loadClass('Zend_Controller_Action');
Zend::loadClass('Zend_View');
class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
    }
    public function noRouteAction()
    {
        $this->_redirect('/');
        echo 'norouteaction';
    }
    public function voteAction()
```

```

    {
        $db = Zend::registry('db');
        $view = Zend::registry('view');
        $view → title = 'Please Vote for this week's movie!';
        $view → sum = $db → showVote();
        $view → result = $db → showData();
        echo $view → render('vote.php');
    }
}
?>

```

We would like to display the survey results currently in the database on the main page. To do that, we created a `voteAction()` method, where we initialized the database and view registries. Then, we call a database method `db → showVote()` which queries the `movie_names` table and gets the sum of the votes. Then, we used the sum to create the percentage of the votes and called another database method `db → showData()` to get the movie names. Finally, we rendered the page to display the movie names along with the percentage of votes. We also allowed users to add a new movie name and vote for the new suggestions. We handled the database insert for the new movie suggestion in `AddController.php`. We created a `movieAction()`, which uses `dbAdapter` method `insert()` to add movie into the `db2` table. The code in Example 3-9 shows `AddController.php`. The `movieAction()` method calls database functions to perform insertions and updates.

Example 3-9 AddController.php

```

<?php
Zend::loadClass('Zend_Controller_Action');
class AddController extends Zend_Controller_Action
{
    function indexAction()
    {
        $this → _redirect('/');
    }
function __call($action, $arguments)
{
    $this → _redirect('/');
}
function movieAction()
{
    $view → title = 'Name your favorite movie of the week!';
    $movie_name = $_POST['movie_name'];
    $votevalue = $_POST['vote'];
    $db = Zend::registry('db');

```

```

        $view = Zend::registry('view');
        if ($movie_name)
            $db → addMovie($movie_name);
        if ($votevalue)
            $db → updateVote($votevalue);
        $view → sum = $db → showVote();
        $view → result = $db → showData();
        echo $view → render('vote.php');
    }
}
?>

```

Note: `indexAction()` in the controllers other than `indexController` should never be called. Because you do not have any control over the user being creative and modifying the URL to access anything, you should always redirect the URL appropriately. Also, in order to support dynamic actions, such as `add/123`, you must implement `__call()` functions.

3.4.1 Integrating with databases: Zend_Db_Adapter

`Zend_Db_Adapter` is the database API abstraction layer for the Zend Framework. You can use same `Zend_Db_Adapter` to connect to and work with many databases, including MySQL, DB2, Sybase, SQL Server, and others. You first need to create an instance of your database server. The code shown in Example 3-10 shows how to create the instance of DB2 database.

Example 3-10 Creating instance of DB2 database using Zend_Db_Adapter

```

$params = array('host' => '9.30.76.201',
                'port' => '50000',
                'username' => 'db2admin',
                'password' => '*****',
                'dbname' => 'movie');

$conn = Zend_Db::factory('Db2', $params);

```

You can ignore the host and port parameters if the DB2 server is local to the framework. Once a DB2 instance is created, it can be passed to the database class being initialized. For our sample application, we create a Database class `database.php`. The code shown in Example 3-11 on page 112 creates an instance of the database class and makes our database object available through the registry variable. A registry is a Zend framework object that can save

information (for example, a database connection or view) and be made available in your Web application when required.

Example 3-11 Registering database

```
$db = new Database($conn);  
Zend::register('db', $db);
```

For a database driven application, the database registration code as shown in Example 3-11 should be placed in the bootstrap file located under framework's www directory. Once registered, the database object can be accessed anywhere just by making it available through the registry:

```
$db = Zend::registry('db');
```

Now, our simple database class implements the utility methods to add, update, and query the movie database as shown in Example 3-12.

Example 3-12 Database class

```
<?php  
class Database  
{  
    private $_db;  
  
    public function __construct($conn)  
    {  
        $this → _db = $conn;  
    }  
  
    public function addMovie($movie_name)  
    {  
        $row = array('name' => $movie_name);  
        $table = 'movie_names';  
        $rows_affected = $this → _db → insert($table, $row);  
    }  
  
    public function updateVote($vote)  
    {  
  
        $sql = "UPDATE movie_names set vote = vote+1 where name='$vote'";  
        $this → _db → query($sql);  
    }  
  
    public function showVote()  
    {  
        $sql = "SELECT sum(vote) as sum from movie_names";  
        return $this → _db → fetchRow($sql);  
    }  
}
```



```

    }
    public function showData()
    {
        $sql = "SELECT * from movie_names order by vote DESC";
        return $this → _db → fetchAll($sql);
    }
}
?>

```

The important part of the database class in Example 3-12 on page 112 is the constructor function, which expects the instance of the db adapter class. It assumes that the database and tables exist already. We have created our movie table manually, but you can certainly construct the `CREATE TABLE` statement and use framework's `direct query()` function to create the table for the first time.

As you can see in the `Database.php` we have used the `fetchRow()` function in the `showVote()` method and the `fetchAll()` function in the `showData()` method. Zend Framework provides the `fetch*()` series of methods, which executes the supplied query and process result. You can also pass the placeholder and an array of bind values to be quoted and replaced into the statement for you. These are the `fetch()` methods available for you to use:

- ▶ `fetchAll()`
- ▶ `fetchAssoc()`
- ▶ `fetchCol()`
- ▶ `fetchOne()`
- ▶ `fetchPairs()`
- ▶ `fetchRow()`

The framework user manual shows sample code snippets on their usage. In addition to the convenience methods we have already discussed, the framework also provides some tools, which help you construct SQL statements and manipulate database objects:

- ▶ `Zend_Db_Select`

This tool provides built-in functions to construct the `SELECT` statement with various clauses. Look at Example 3-13.

Example 3-13 Constructing SELECT using Zend_Db_Select

```

<?php
//
// SELECT *
// FROM movie_names
// WHERE vote > 2

```

```

// ORDER BY name
$select = $db->select() // to create object
// you can use an iterative style...
$select → from('movie_names', '*');
$select → where('vote > ?', 2);
$select → order('name');

// ...or object chaining:
$select → from('movie_names', '*');
        → where('vote > ?', 2);
        → order('name');

// regardless, fetch the results
$sql = $select → __toString();
$result = $db → fetchAll($sql);

// alternatively, you can pass the $select object itself;
// Zend_Db_Adapter is smart enough to call __toString() on the
// Zend_Db_Select objects to get the query string.
$result = $db → fetchAll($select);
?>

```

► Zend_Db_Table

It is a `Zend_Db_Adapter` class, which lets you examine the table schema and then aids you in manipulating and fetching rows from the table. You need to create a class with the same name as the table name (camelized), which extends `Zend_Db_Table` class. By default, the camelized class name maps to the underscored table name, but you can also override this by redefining the `$_name` property.

For more information, refer to the Zend online documentation at:

<http://framework.zend.com/manual/en/zend.db.table.html>

Note: The database components of the Zend Framework are relatively unstable and new features and functionalities are being added. Some of the database object features might not work in the preview release, but the good news is that you are free to manipulate the code and design the function your own way. For example, we have used the adapter's `insert()` method to add movie names to the database. Alternatively, your application can construct the `INSERT` statement and use adapter's `direct query()` function to execute and achieve the same result. The actual implementation of these API methods available for your use are in `Db2.php` located under the `lib/Zend/Db/Adapter` directory.

Integrating the components

To put this all together, we create database.php in lib directory. The __autoload() function in our bootstrap file, www/index.php, is able to load it when needed. The bootstrap file also instantiates \$db and \$view and loads them in registry variables. Finally, we create a template to display the result. In the case of our small survey application, we create an HTML template embedded with some php code. Typically, we recommend that you create different templates for different URLs supported by the application, and one index.php exists as an initial template. vote.php is our only template and displays both initial movie information and modified survey results, because this is the most appropriate way of displaying results for this application. The code in Example 3-14 shows the app/view/vote.php.

Example 3-14 vote.php

```
<html><head>
  <title><?php echo $this → escape($this → title); ?></title>
</head><body>
<h1>Name your favorite movie of the week</h1>
<form action="/add/movie" method="POST">
  Suggestion: <input type="text" name="movie_name"><P>
  <INPUT type="submit" value="Submit new movie and/or vote">
<?php
$movie_name = $_POST['movie_name'];
$vote = $_POST['vote'];
  echo "<table border=0><tr><th>Vote</th>";
  echo "<th>Idea</th><th colspan=2>Votes</th>";
  echo "</tr>\n";
$sum = $this → escape($this → sum['SUM']);
foreach ($this → result as $row) { ?>
<TR><TD align=center>
<input type="radio" name="vote" value='<?php echo
$row['NAME'];?>'></td><td>
<?php
  echo $row['NAME']. "</td><td ALIGN=right>";
  echo $row['VOTE']. "</td><td>";
  if($sum && (int)$row['VOTE']) {
    $per = (int)(100 * $row['VOTE']/$sum);
    echo "<img src=/images/logo.jpg height=12 ";
    echo "width=$per> $per %</TD>";
  }
  echo "</tr>\n";
}
echo "</table>\n";
?>
```

```




---



```

With everything in place, you can see that the presentation layer is separate from the business logic layer, and adding more business logic and presentation is extremely easy. The code looks clean; it is flexible and extensible. The end result is the Web site shown in Figure 3-2.

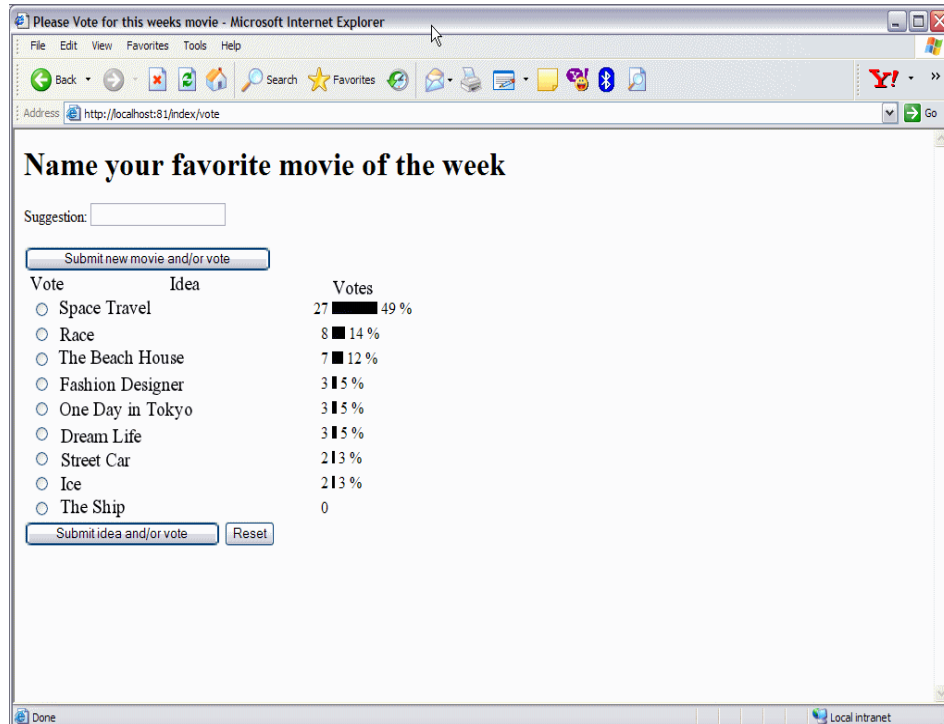


Figure 3-2 Movie of the Week initial page

3.4.2 Zend framework: XCS

One of the major advantages of using a Web framework is that it provides the abstraction to the database layer to create a database driven Web site easily. The developer can concentrate on the behavior of data rather than the details of database access and manipulation. With the amount of proliferation of XML data over the Web and elsewhere, there is a need to abstract the persistence access of XML data. DB2 Express-C V9 provides pureXML storage to store and manage XML data in its native structure. XML Content Store (XCS) has been created to help simplify the development of XML-centric Web applications. XCS is an

incubator technology in Zend Framework and provides both a persistence data access layer as well as an API for managing XML data easily.

In this section, we provide an architectural overview of the XCS. We describe each of the components that makes up the XCS, and we explain how they work together. Figure 3-3 illustrates the XCS architecture.

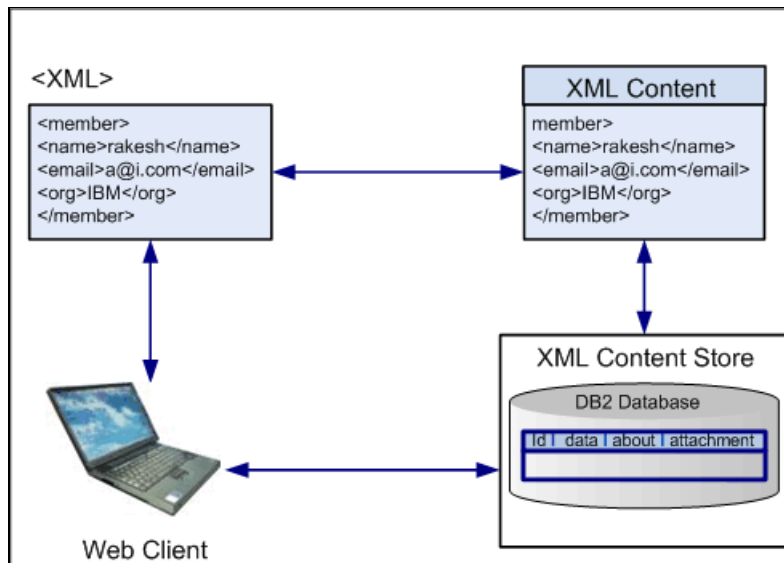


Figure 3-3 XCS architecture overview

XMLContent (Zend_Db_Xml_Content)

As a developer, you can encounter XML data in many forms, such as Web services messages, RSS/Atom feeds, and configuration files. Once you decide that this XML data needs to be saved somewhere, we can assume several things about this data:

- ▶ First, we need a way to uniquely identify it, so that once it is saved, it can easily be programmatically found and retrieved. The unique name can be a numeric ID or a user-provided name.
- ▶ The second assumption is that the XML data will be stored “as is”. It will not be modified or changed in any way. Any modifications that are required will be made by the application outside of the XCS. Internally, the XML data is stored as a DOM document, but an application is free to access the data as a file stream, string, or several other convenient access methods which can or cannot be implementation-dependent.
- ▶ Third, the capability is provided to add additional metadata about the XML data that is saved. For example, if the XML data is a blog entry, perhaps the

application would care to know the date and title of the entry, or the hostname where the entry originated. This metadata is saved in an about property and is also XML.

- ▶ Finally, often XML data is accompanied by binary data, such as .jpeg, .pdf, or .doc files. An attachment property associates this binary data with the XML data. In the current implementation of `Zend_Db_Xml_Content`, the attachment property can contain either zero or at most one item, though optionally, a future version can contain any number of items.

These properties of the XML data are encapsulated in an object called *Zend_Db_Xml_Content*. `Zend_Db_Xml_Content` objects are the fundamental components of the XCS because they are the XCS representation of XML data. As we will see, the `Zend_Db_Xml_ContentStore` component needs to know about the persistence technology (for example, a relational database) and how to access it, but `Zend_Db_Xml_Content` objects do not need to know anything about it.

Zend_Db_Xml_ContentStore

Zend_Db_Xml_ContentStore is an abstract class that represents a repository of XML documents. It is responsible for updating the data source based on changes made to a `Zend_Db_Xml_Content` object in the repository as well as retrieving `Zend_Db_Xml_Content` objects from the data source based on search or ID criteria. A *data source* is defined very generally as the persistence layer where the XML data is saved. It can be a relational database, an XML database, or a file system, and it stores the XML data in its own format. When an `Zend_Db_Xml_ContentStore` object is instantiated, it receives a *connection handle*, which describes in a meaningful way what the data source is.

Example 3-15 shows how to create an XML ContentStore using DB2 dbAdapter.

Example 3-15 Creating an XML ContentStore

```
// Create a DBAdapter using database connection properties
$dbuser = 'db2admin';
$dbpass = 'db2admin';
$dbname = 'contacts';

$params = array('username' => $dbuser,
               'password' => $dbpass,
               'dbname'   => $$dbname);

$conn = Zend_Db::factory('Db2', $params);
$xcs = new Zend_Db_Xml_XmlContentStore_Db2($conn, 'blogs');
```

In Example 3-15 on page 118, XML ContentStore \$xcs has been given a name, so \$xcs only manages XML data related to specific application *blogs*. Other content stores can be created to manage other types of XML data.

Zend_Db_Xml_ContentStore knows how to access the persistence layer to perform Create, Read, Update, and Delete (CRUD) operations and abstracts these operations by using convenience methods. For example, if the persistence layer is a relational database, a call to the method insert() builds an appropriate SQL insert statement based on the structure of the underlying tables used and the content of the Zend_Db_Xml_Content object. It connects to the database and executes the statement. Other CRUD methods work in a similar fashion and include update(), delete(), deleteById(), and selectAll(). Example 3-16 shows how to save XML data in the XML ContentStore.

Example 3-16 Saving XML data in the XML ContentStore

```
// create an Zend_Db_Xml_Content object from a file
// using a utility method
$xmlDoc = Zend_Db_Xml_XmlUtil::createDocument();
$xmlDoc = Zend_Db_Xml_XmlUtil::importXmlFile('data.xml');

//save the XML document to the persistence layer

$xcs → insert($xmlDoc);
```

Similar functions exist for update and delete. In addition, Zend_Db_Xml_ContentStore also contains a simple search facility that retrieves Zend_Db_Xml_Content by its ID or by searching within the XML data or the metadata in Zend_Db_Xml_Content. The search on XML data is done using XPath expressions. Some of these methods are find() and findById().

Zend_Db_Xml_ContentStore_DB2

The class XMLContentStoreDB2 is implemented using the DB2 adapter which in turn uses the ibm_db2 CLI driver. For example, to insert, the application gives the XCS an XMLContent object and the XMLContentStoreDB2 takes care of the underlying details. Because DB2 9 supports a native XML data type, one Zend_Db_Xml_ContentStore_DB2 object maps to one table with four columns named as ID, DATA, ABOUT, and ATTACHMENT. The table columns are defined as follows:

- ▶ ID is a unique integer and is used as the primary key of the table.
- ▶ DATA is defined as an XML column.
- ▶ ABOUT is defined as an XML column.
- ▶ ATTACHMENT is defined as a BLOB column.

Using DB2, each row in the table represents one `Zend_Db_Xml_XmlContent` object.

Zend_Db_Xml_XmlIterator

It is possible that a search returns one `Zend_Db_Xml_Content` or a set of `Zend_Db_Xml_Content` objects. In the case where a set is returned, the `Zend_Db_Xml_XmlIterator` class is used to iterate over the set of XML documents that meets the search criteria. `Zend_Db_Xml_XmlIterator` implements the `Iterator` interface so it knows several essential things about the set of `Zend_Db_Xml_Content` objects over which it is iterating. These include its current location in the set, how to retrieve the next object in the set, how to go back to the beginning of the set, and when it has reached the last item in the set. This allows the developer to assign behavior on the XML data at each iteration without having to worry about the details of loop control. An example of `XMLIterator` is shown along with the example of other utility functions in the next section.

Zend_Db_Xml_XmlUtil

`Zend_Db_Xml_XmlUtil` is a utility class that provides static convenience methods for passing XML data back and forth from the application to `Zend_Db_Xml_XmlContent`, either for the raw XML data or for the about XML metadata. Though the XML is stored internally as a DOM, `Zend_Db_Xml_XmlUtil` can allow an application to use strings, file streams, or any other implementation-specific object representation of XML data, such as PHP's SimpleXML. Convenience methods for converting between these different types of representations and DOM are provided. Let us assume the following document is already saved in the persistence layer as shown in Example 3-17.

Example 3-17 Sample XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <title>Mom, Dad, and The Rainbow House</title>
  <author>Mary Staple</author>
  <date>2006-05-09</date>
  <body>Another fantastic children's book from bestseller Mary Staple.
</body>
</book>
```

To search anywhere for the keyword "Rainbow", see Example 3-18.

Example 3-18 Searching keyword

```
keyword = 'Rainbow';
```



```
$found = $xcs → findAnywhere($keyword, Zend_Db_Xml_XmlUtil::DATA);
```

To be more specific about case and where exactly in the document to search, see Example 3-19.

Example 3-19 Detail searching

```
$search = array('title' => 'Rainbow',
               'author' => 'Staple');

$options = array('caseSensitive' => true,
                'exactMatch'     => false,
                'logic'          => 'or');

$found = $xcs → find($search,
                    Zend_Db_Xml_XmlUtil::DATA,
                    $options);
```

find() returns an XML iterator, which represents a set of XML documents as shown in Example 3-20.

Example 3-20 Get XML iterator

```
foreach($found as $current) {
// get a SimpleXML object and do something with it
$xml = Zend_Db_Xml_XmlUtil::exportToSimpleXML(
    $current,
    Zend_Db_Xml_XmlUtil::DATA);
// prints the contents of the <title> element
echo $xml → title;
}
```

We can also use a simple XPath expression to search as shown in Example 3-21.

Example 3-21 Search using XPath expression

```
$xml = $xcs → executeXPathPredicateQuery("/book[date='2004-12-21']",
    Zend_Db_Xml_XmlUtil::DATA);
foreach ($xml as $current) {
    $xml = Zend_Db_Xml_XmlUtil::exportToSimpleXML($current,
    Zend_Db_Xml_XmlUtil::DATA);
    echo '<p>Title: ' . $xml → title;
}
```

This will output Mom, Dad, and The Rainbow house.

Parameters are also accepted in XPath searches using an associative array. It is important to match XPath variable names with the keys of the associative array. Otherwise, the XPath search fails. Example 3-22 shows the XPath search.

Example 3-22 XPath search

```
$param = array();
$params['d'] = '2006-05-09';
$params['t'] = 'Mom, Dad, and the Rainbow House';

$xml = $xcs → executeXPathPredicateQuery(
"/book[date=\$d or title=\$t]", $params);
foreach ($xml as $current) {
    $sxml = Zend_Db_Xml_XmlUtil::exportToSimpleXML($current,
Zend_Db_Xml_XmlUtil::DATA);
    echo '<p>Author: ' . $sxml → author;
}
```

This will output “Author: Mary Staple”.

3.4.3 myContacts.com: An XCS application

In any application environment, there are two types of data used, data for persistent storage and data for message flows. XML is undoubtedly suited for both types of data: XML for messages and XML for persistent storage of data. XML for messages possesses fewer design problems, because each message is self-contained and the application process usually decides what should be included in a message. Contrary to this, while designing the persistent XML data, its static model is very important. The real design challenge is to decide the granularity of data and the size of the XML documents and whether to keep everything in one large document or create a large number of small documents. Irrespective of the size of the XML data, finding information is always a two step process, finding the right document and then finding the desired information within the document. The architecture of XCS makes the XML persistence model easy. Several types of applications benefit from XCS.

Here are few examples:

- ▶ RSS Feed/Blogging Application
Users can specify feeds, applications can pull feed data and store it in XCS. Applications can then search and display feeds. Blogs can also be created and stored in XCS. Blogs then can be searched, sorted, and displayed.
- ▶ XML-based Content Management system
All types of content can be used to store data in the XCS DATA column. ATTACHMENT can be used to store binary objects such as image data. ABOUT can store the metadata which can drive the application process flow.
- ▶ Web services
Web services are the applications that utilize XML messages for interaction between components. Applications can utilize XCS to store, search, and retrieve XML data.

Essentially, any application that requires XML data interchange can benefit from XCS. We have created one application to demonstrate the capability of XCS. Our sample application is a social network application. We named it myContacts.com.

This application enables users to:

- ▶ Create and maintain a profile for social and professional networking.
- ▶ Search for people within networks.
- ▶ Make contacts with the people and collaborate with them.

Solution components

This application can be divided into the following functional components:

- ▶ Create and maintain member profile
We need a persistent XML storage to store all of our member profiles and relationships between them. We also need to build user interfaces for new members to create profiles and log in to the system. We need to track user inputs, build XML with it, and then store it in XCS.
- ▶ Track logged in users
We need to track logged in users so that the application knows if a user is viewing the user's own profile or another user's profile. This is required to allow users to make contact with other members. We used PHP's built-in session tracking methods to track the status of logged-in users.
- ▶ Update contacts and relationships
We need user interfaces and controller functions to perform the updates on XML data. Luckily, we have XCS implementations of Create, Remove,

Update, and Delete (CRUD), so we do not need to worry about underlying database calls. We use XMLContent's ABOUT property to store members' relationships.

► Search

We need to implement search, so that users can look for members they want to contact. We utilize XCS's find functions to implement the search without knowing any XQuery implementation programming.

Solution overview

Figure 3-4 shows the navigation path of the proposed application. In the framework design, the Web page navigation flow should decide how the Controllers and Views should be created.

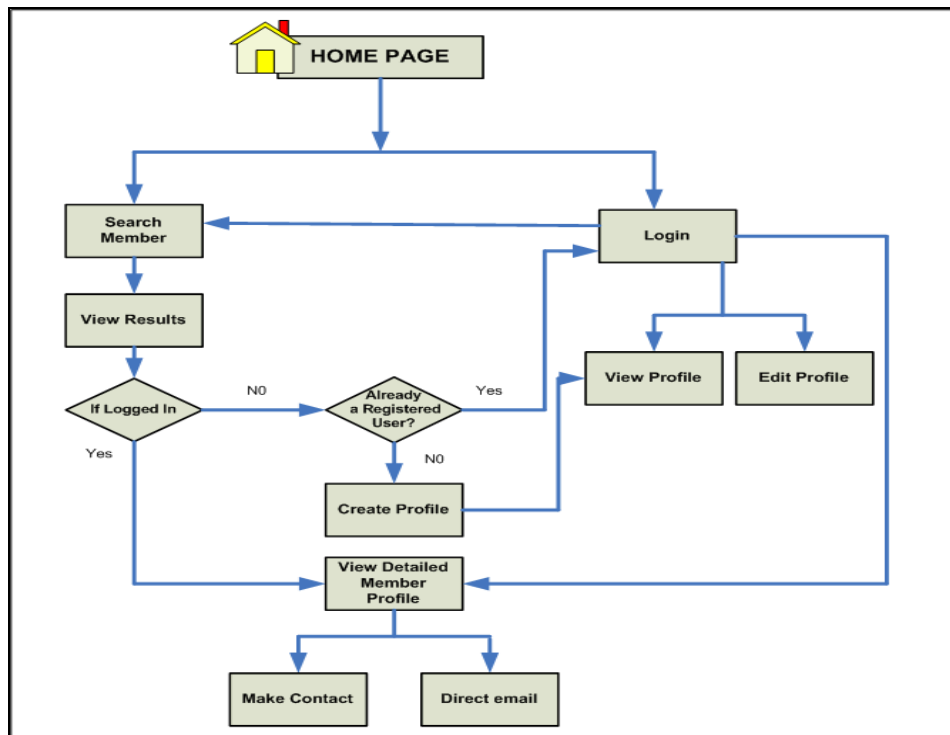


Figure 3-4 Navigation diagram for MyContacts.com application

As you can see, we give the user a choice of options. The user can either log in or search for someone who is already using this Web site. A user can search for a member, but the user cannot view a member's detailed profile until the user logs in. If the user is new, the user is able to create and save the user's profile in the system. We also give the user the option to find a particular member and

make contact with a user-selected relationship. The user also is able to contact others directly via e-mail. The member's contact information is displayed along with the profile.

A summary of the files we use in the myContacts application is shown in Table 3-2.

Table 3-2 Files used in myContacts

Name	Type	Description
index.php	bootstrap	Any call to the main URL calls this bootstrap file
style1.css	CSS	Style sheet for HTML presentation
IndexController.php	Controller	Main controller
AddController.php	Controller	Contains add/delete/update actions
ViewController.php	Controller	Contains navigation/output functions
index.php	View	Initial view of the application
member.php	View	Displays member profile data
searchResult.php	View	Displays search result
view.php	View	Interface to get new user profile data
thanks.php	View	Acknowledgement of successful user action
error.php	View	Displays error messages
Database.php	Adapter	Contains database functions

Let us go ahead and look at the application.

Setting up the database

The database for the myContacts application is fairly simple, because users are able to add new members and relationships using the application itself. We need to store user names and passwords in a DB2 table DB2ADMIN.MEMBER. We also need to create a table required by XCS. You can set up the database for this application by running the SQL on the DB2 Command Line as shown in Example 3-23.

Example 3-23 Database setup script

```
create database contacts using codeset utf-8 territory us;
connect to contacts;
create table db2admin.member(xmlid BIGINT, email VARCHAR(50) NOT NULL
PRIMARY KEY, passwd VARCHAR(10), fname VARCHAR(30), lname VARCHAR(30));
```

```
db2 create table db2admin.xmldata(  
    id BIGINT NOT NULL generated always as identity primary key,  
    data XML,  
    attachment BLOB(100m),  
    about XML);
```

The XML data type is available on any UTF-8 database in the DB2 9 release. With the creation of these database objects, DB2 is now ready for use as an XML Content Store. There are no other DB2 administrative tasks necessary. However, to improve performance when searching, we suggest that you create indexes for the XML data as shown in Example 3-24.

Example 3-24 Creating index script

```
create index datatext on xmldata(data) generate key using xmlpattern  
'//*' as sql varchar(800)  
create index dataattr on xmldata(data) generate key using xmlpattern  
'//@*' as sql varchar(800)  
  
create index abouttext on xmldata(about) generate key using xmlpattern  
'//*' as sql varchar(800)  
create index aboutattr on xmldata(about) generate key using xmlpattern  
'//@*' as sql varchar(800)
```

These are very general indexes on all text nodes and attributes on both DATA and ABOUT. The developer might decide to create indexes on elements or attributes for application-specific data, such as a product ID or a last name element where searches might occur often.

Remember that you can run these commands individually or run the script `setupdb.sql`:

```
db2 -tvf setupdb.sql
```

Tip: The XCS table creation is optional. The XCS is capable of creating the table and indexes on first use.

This script is also part of the zip file for the entire application which is downloadable at the redbook Web site.

Getting Started

When a user loads the URL `http://localhost/`, the user sees the page shown in Figure 3-5.

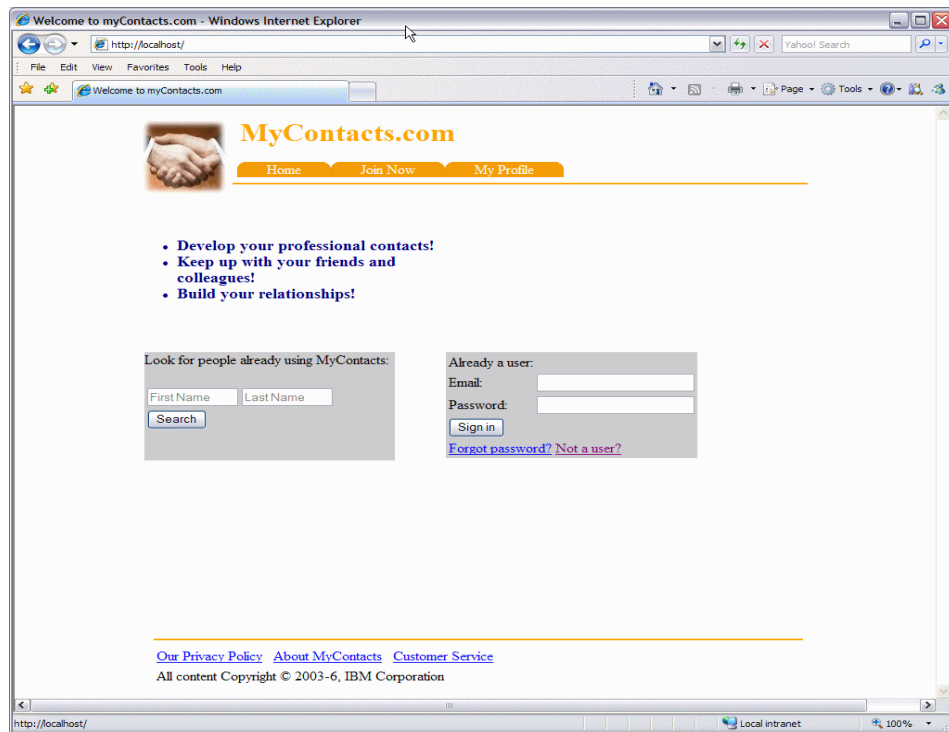


Figure 3-5 MyContacts.com Index page

Controllers

MyContacts.com has the following controllers:

- ▶ IndexController

Our IndexController is as clean as it should be. It has a `indexAction()` method to serve the main Web page (Home page) and a `noRouteAction()` method to reroute users to the home page in case users become creative and modify the links. Example 3-25 shows the IndexController.php.

Example 3-25 MyContacts.com: IndexController.php

```
<?php
class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
```

```

        /* show the login page. */
        $db = Zend::registry('db');
        $view = Zend::registry('view');
        $view → title = 'Welcome to myContacts.com';
        echo $view → render('index.php');
    }
    public function noRouteAction()
    {
        $this → _redirect('/');
        echo 'norouteaction';
    }
}

```

When a user tries to log in, a database query is issued and the member's profile displays. If the member ID is found and the password is validated, the session variable is updated with the user's ID. The view/member action is invoked and the output looks like Figure 3-6.



Figure 3-6 My Profile page showing logged-in user and his contacts

▶ AddController

Our AddController contains the following action methods (we discuss these action methods later to explain important concepts):

- indexAction() to reroute users to the home page
- memberAction() to add new member data
- contactAction() to add a relationship between members
- __call() to handle the calls not defined by any action methods

▶ viewController

Our viewController contains the following action methods:

- indexAction() to reroute users to the home page
- searchAction() to implement existing member searches
- displayAction() to present the detailed member profile
- __call() to handle the calls not defined by any action methods such as other database action controls

Views

The application has the following major views:

▶ index.php

This template presents the home page with two interfaces. The search allows users to find members already in the database. The login window is also provided on this page.

▶ member.php

This template presents the logged in member's profile data and also a searched member's profile data. This also presents the relationship information under the member's contacts or my contacts.

▶ searchResult.php

This template displays the search result for a member. The return search result is clickable to get the detailed profile, if the user is logged in.

▶ view.php

This template provides the interface to the user to create a new profile.

The other views, such as thanks.php and error.php, assist users with useful information. There might also exist some views to display the user agreement, privacy policy, customer support information, and other relevant information as shown by the link in the footer.

Note: These might not be necessarily implemented in the code either because they are not important conceptually or beyond the scope of this chapter.

Database class

We created a database class, Database.php, where we implemented the following functions:

► `__construct()`

This is the database class constructor function to initialize both the dbAdapter connection object and the XMLContentStore connection object.

Note: \$conn is the connection object of the DB2 database we created in our bootstrap file `www/index.php`.

Example 3-26 shows `__construct()`.

Example 3-26 Database class: constructor method

```
private $_db;
private $_db1;
public function __construct($conn)
{
    $this->_db = $conn; // for dbAdapter
    // for XML Content store
    $this->_db1 = new Zend_Db_Xml_XmlContentStore_Db2($conn);
}
```

► `login()`

This method provides the SQL query to go against the member database to check if the user profile exists. Also, it returns a member's ID so that the member's detailed profile can be fetched from XCS. Example 3-27 shows the `login()` method.

Example 3-27 Database class: login method

```
public function login($email, $passwd)
{
    $sql = "SELECT xmlid, fname, lname from member where email=
? and passwd= ?";
    $param = array(); // an array is created for parameters
    $param[] = $email;
    $param[] = $passwd;
    if ($result = $this->_db->fetchAssoc($sql, $param)) {
        return $result;
    }
    return FALSE;
}
```

► `getMember()`

This method implements a database convenience method to retrieve member information from the member table when given the member ID. It uses the simple `dbAdapter` function to fetch data in a similar way to the login function.

► `search()`

This method provides some validation and posts an SQL/XML query to get a member's profile. It provides fuzzy search capability on first and last names. As you can see, we needed to join on the `MEMBER` table and `XMLDATA` and use SQL/XML function to get the search result we want. Example 3-28 shows an example of the flexibility of the XCS. If the existing API methods are not sufficient for your needs, you can take control of the database by writing your own SQL queries, SQL/XML queries, or XQueries directly.

Example 3-28 Database class: search method

```
public function search($fname, $lname)
{
    $sql = "SELECT m.xmlid, m.fname, m.lname, m.email,";
    $sql .= "xmlserialize(xmlquery('\$data/member/org/text()' ";
    $sql .= "passing x.data as \"data\") as varchar(120)) as
company,";
    $sql .= "xmlserialize(xmlquery('\$data/member/title/text()' ";
    $sql .= "passing x.data as \"data\") as varchar(120)) as title ";
    $sql .= "from db2admin.member m, db2admin.xmldata x where ";
    if ($fname && $lname) {
        $param = array();
        $sql .= "(upper(m.fname) like ? or upper(m.lname) like ?)
AND ";
        $param[] = strtoupper("%$fname%");
        $param[] = strtoupper("%$lname%");
    } else if ($fname && !$lname) {
        $param = array();
        $sql .= "(upper(m.fname) like ?) AND ";
        $param[] = strtoupper("%$fname%");
    } else if (!$fname && $lname) {
        $param = array();
        $sql .= "(upper(m.lname) like ?) AND ";
        $param[] = strtoupper("%$lname%");
    }
    $sql .= "m.xmlid = x.id";
    if ($param && $result = $this → _db → fetchAssoc($sql,
$param)) {
        return $result;
    }
}
```

```
return false;
}
```

This application allows users to create and save their profiles. A user interface is provided as shown in Figure 3-7. Once a user fills in the required text and clicks **Join Now**, a new member profile is added into the DOC column of the XCS database. This profile is saved as XML. An empty contact data of XML type is also constructed for this profile. When this user updates the user's contacts, this XML document is updated. With the new profile created, the authentication table is also updated.

Welcome to myContacts.com - Windows Internet Explorer
http://localhost/add/new
Welcome to myContacts.com
MyContacts.com
Home Join Now My Profile
Create new member profile:
First Name: John
Last Name: King
Email Address: jking@gmail.com
Password (6 or more chars):
Re-enter Password:
City: Minneapolis
State: Minnesota
ZIP Code: 55418
Company/Organization: Ameriprise Financial
Title: Financial Advisor
Industry: Biotechnology
Education(Optional): Masters Degree
Present & Past Experiences: I am an Investment Expert in the field of biotechnology incubators. I am a MBA from RICE and a five start Ameriprise financial expert!
Join Now
By clicking "Join Now", you are indicating that you have read, understood, and agree to MyContacts's User Agreement and Privacy Policy.

Figure 3-7 Create a new member profile

► addMember()

This method adds member data into the relational table when a new member profile is created. The member ID is auto-generated for XML detailed profile data, which is, in turn, used for inserting into the member table. Example 3-29 on page 133 shows the addMember() function.

Example 3-29 Database class: addMember method

```
public function addMember($xmlid, $email, $passwd, $fname, $lname)
{
    $row = array (
        'xmlid'=> $xmlid,
        'email'    => $email,
        'passwd' => $passwd,
        'fname'=> $fname,
        'lname'=> $lname
    );
    $table = 'member';
    // insert the row and get the row ID
    $rows_affected = $this → _db → insert($table, $row);
}
```

► saveNew()

This method inserts a new XML record into XMLDATA, using XCS's convenience method insert(). Example 3-30 shows the saveNew() function.

Example 3-30 Database class: Insert profile

```
public function saveNew($entry)
{
    return $this → _db1 → insert($entry);
}
```

► getProfile()

This method uses XCS's convenience method findById() that takes docid as input and returns the XML record as an array. Example 3-31 shows the getProfile() function.

Example 3-31 Database class: Retrieve profile

```
public function getProfile($xmlid)
{
    $result = $this → _db1 → findById($xmlid);

    if(!is_null($result)) {
        return $result;
    }
    return false;
}
```

The Add Contact/relationship feature allows a member to make contacts with other members and create the chain of professional relationships. The user

must log in to view a member's detailed profile and make the member a contact. The user can select a relationship name from the drop-down box. When the user clicks **Make Contact**, a XCS update is issued to update XML data in the about column which holds the member contact data. If a duplicate relationship with the same person is found in the database, the application ignores the request and does not make the update. If a user tries to add the user as a contact, an error message is thrown and database update does not occur. Finally, if there is no error, database update succeeds and the user is taken to that user's own profile page which reflects the currently added contacts. Alternatively, a logged-in user can at any time, click the **My Profile** menu bar to view that user's own profile and contact information.

Figure 3-8 shows the Web page where a user can define a relationship and make contacts as we discussed.

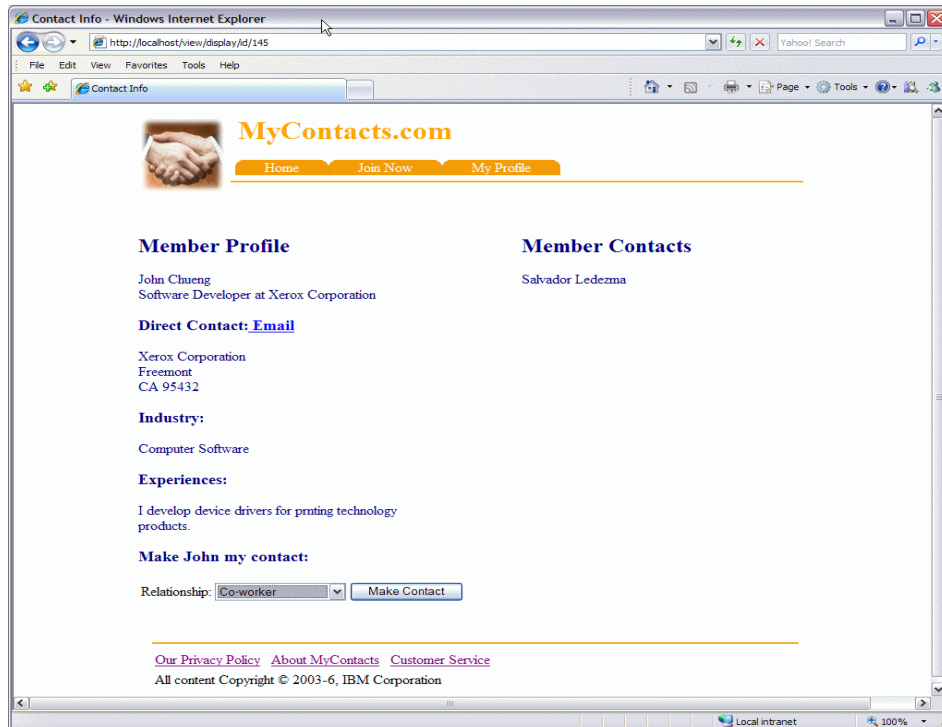


Figure 3-8 Define relationship and make contact

► addContact()

This implements a method to add a contact/relationship with an existing myContacts member. From the application point of view, a logged-in member can search for other members by first name and/or last name and then view

another member's detailed profile. The authorized user can add the result member as a contact. It is worth explaining here how XCS makes creation and manipulation of xml data easier. XCS's `Zend_Db_Xml_XmlUtil` class provides a convenience method to create a simpleXML representation of the DOM object, which is easy to handle programmatically. Let us discuss the flow here:

- We get XML data using XCS's `findById()` method, passing the `$id` of authorized user as shown in Example 3-32.

Example 3-32 XCS convenience method findById()

```
$result = $this → _dbl → findById($id);  
$xmlContent = $result → current();
```

- `xmlContent` object now has the XML data. Remember our XML data consists of two XML columns and two non-XML columns. Using the XCS accessor's method, you can access the data or about contents easily as shown in Example 3-33, the first parameter is input DOM object and the second parameter is what to retrieve.

Example 3-33 XCS convenience method exportToSimpleXml()

```
$xml = Zend_Db_Xml_XmlUtil::exportToSimpleXML($xmlContent,  
Zend_Db_Xml_XmlUtil::ABOUT);
```

- Now, since you have a simpleXML representation of your DOM data, you can use any XPath expression to traverse into it and create an array to access the data elements. In this application, we prohibit a user from creating duplicate relationships with the same member and also from creating any relationship with the actual user. Example 3-34 shows how to build XML data for contact.

Example 3-34 Building XML data for contact

```
foreach ($xml → xpath('//entry') as $curr) {  
    if ((int)$curr['id'] == $idToAdd) {  
        $relExists = false;  
        foreach ($curr → relationship as $currRel) {  
            if ((string) $currRel == $relationship) {  
                $relExists = true;  
                $exists = true;  
                break;  
            }  
        }  
    }  
    if (!$relExists) {  
        $curr → addChild('relationship');  
        $curr → relationship[++$count] = $relationship;  
    }  
}
```

```

    $exists = true;
    break;
}
if ($exists) {
    break;
}
}
}
$count++;
}

```

-
- When we are ready to manipulate the XML and put it back into the `xmlContent` store, we can simply use XCS's `importSimpleXML()` method to build the document and the call `update()` method to store it in XCS, as shown in Example 3-35.

Example 3-35 XCS update

```

$entry = $xml → addChild('entry');
$entry → addAttribute('id', $idToAdd);
$entry → addChild('relationship');
$entry → relationship = $relationship;
$xmlContent → about =
Zend_Db_Xml_XmlUtil::importSimpleXML($xml);
$this → _db1 → update($xmlContent);

```

► `processContact()`

We created the method to get contact/relationship information from XCS and keep in an array variable for easy manipulation and display on the form. It also uses the `exportToSimpleXml()` convenience method.

Add Controller

Following the framework coding convention, we created controller functions related to update and insert into `addController.php`. We walk you through the important concepts, while discussing the action methods of `addController.php`.

In any Web application processing, the user inputs via Web pages and validating them for further processing is not only important but also cumbersome. We do this in the `memberAction()` method of `addController`. `Zend_Filter_Input` is a framework component, which provides simple facilities to promote a structured and rigid approach to input filtering of user data. PHP developers use various types of input filtering techniques, including whitelist filtering, blacklist filtering, regular expressions, conditional statements, and so on, but the good thing about `Zend_Filter_Input` is that it combines all of these techniques into a single API with consistent behavior and strict naming convention. `Zend_Filter_Input` is designed primarily with arrays in mind. Many sources of input are already covered by

PHP's superglobal arrays (`$_GET`, `$_POST`, `$_COOKIE`, and so on.), and arrays are a common construct used to store input from other sources. If you need to filter a scalar, you can use Zend_Filter component. Zend_Filter_Input supports both a strict and a non-strict approach. In a strict approach, a single array of post variable to be processed is passed into the constructor as shown in Example 3-36.

Example 3-36 Zend_Filter_Input

```
$filterPost = new Zend_Filter_Input($_POST);
$email = $filterPost → testEmail('email');
```

Zend_Filter_Input sets the array that is passed (`$_POST`) to NULL, so direct access is no longer possible. If you need to access it directly and do not want Zend_Filter_Input to set the `$_POST` array to null, you can use a non-strict approach and pass FALSE as the second parameter. See Example 3-37.

Example 3-37 Zend_Filter_Input non-strict call

```
$filterPost = new Zend_Filter_Input($_POST, FALSE);
$email = $filterPost → testAlpha('name');
```

Note: The testEmail() function is not available in Zend Framework as of writing this book. You can write your own validation function for e-mail filtering or for our application, we used testAlpha() to validate e-mail. Consider this as an opportunity to get involved with framework community development and contribute to the framework code. Also, for a complete list of available input filtering facilities, refer to the online developers guide at the Zend framework Web site.

Saving Input data into XCS

After we filter and validate the input, we need to save it into persistent store. As shown in Example 3-38, we construct the xml from the `$_POST` array, create a DOM document, create an XMLContent object, and save it into XCS. Depending on your skill level or the need of the application, you can also construct XML using string concatenation or the PHP's simpleXML object. You can add these into XMLContent via an XMLContentUtil method call.

Example 3-38 Saving data to XCS

```
$doc = new DOMDocument();
    $root = $doc → createElement("member");
    $doc → appendChild($root);
    foreach ( $_POST as $key => $value ) {
        $elem = $doc → createElement($key);
```

```
$root → appendChild($elem);
$elemtext = $doc → createTextNode($value);
$elem → appendChild($elemtext);
}
$myDoc = new Zend_Db_Xml_XmlContent($doc);
$about = new DOMDocument();
$abtRoot = $about → createElement("contacts");
$about → appendChild($abtRoot);
$myDoc → about = $about;
$db → saveNew($myDoc);
```

Since the php array objects are associative, that is, stored as a \$key → \$value pair, you can see it is easy to create XML elements and attributes by just looping through the \$_POST array.

Modifying data in XCS

If you notice in the previous example, the add/member action adds a member's profile into data and constructs an empty XML for ABOUT column. An empty element `</contacts>` is created and stored in XCS. When a logged-in user browses a myContacts member and adds the member as a contact, the add/contact action is invoked. The contactAction invokes the database method addContact, which calls XCS's update function to update the entire xml document. The db → addContact() method is explained in detail in the previous database section.

ViewController

Similar to AddController, ViewController must not implement index action. However, this controller should implement a __call() to handle dynamic actions such as the following URL. This makes sure that any URL typing by the user is handled appropriately.

`http://localhost/view/123.`

We discussed other actions such as search and display in detail in the previous database section.

You can download the social network application called myContacts.com application from the IBM Redbooks Web site at:

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

Appendix C, "Additional material" on page 319" describes the download instructions.

3.4.4 Other Zend Framework components

We have discussed so far only a small portion of the functionalities Zend Framework provides. As mentioned previously, the framework fulfills the promises and provides a rich set of components to build today's applications quickly. In this section, we briefly go through some of these components. We recommend that you refer to the Zend online manual for the details and examples.

Zend_Feed

RSS & Atom feeds are hot technologies today and many applications are being built to support them. RSS/Atom is a technology and syndication is a process. Feed is about getting content regularly and sequentially. The core technology is XML on which both are built, and hence, DB2 9 pureXML can be used to store RSS/Atom feeds. XCS makes this job even easier. Zend_Feed provides a natural syntax for accessing elements of feeds, feed attributes, and entry attributes. Zend_Feed also has extensive support for modifying the feed and entry structure with the same natural syntax and turning the result back into XML. In the future, this modification support could provide support for the Atom Publishing Protocol. The Zend online documentation shows an example to demonstrate a simple use case of retrieving an RSS feed and saving relevant portions of the feed data to a simple PHP array, which you could then use for printing the data, storing to a database, and so on.

Zend_Mail

Zend_Mail provides generalized functionality to compose and send both text and MIME-compliant e-mail messages. You can send mail with Zend_Mail via the php built-in mail() function or via direct SMTP connection. Different options can also be used such as sending HTML e-mail or sending attachments with e-mail. A simple e-mail consists of some recipients, a subject, a body, and a sender. To send mail using the Zend_Mail API, follow Example 3-39.

Example 3-39 Using Zend_Mail API to send e-mail

```
<?php
require_once 'Zend/Mail.php';
$mail = new Zend_Mail();
$mail → setBodyText('DB2 Express-C ROCKS with PHP !!');
//to send html email use setBodyHtml() instead of setBodytext()
//the MIME content type will be automatically set to html
$mail → setBodyHtml('<b>DB2 Express-C </b>ROCKS with PHP !!');
//to send attachment use addAttachment()
//by default it assume binary attachment, but second argument
//can override this to image
$mail → addAttachment($someBinaryString);
```

```
$mail → addAttachment($myImage, 'image/gif',  
Zend_Mime::DISPOSITION_INLINE, Zend_Mime::ENCODING_8BIT);  
$mail → setFrom('ranjanr@us.ibm.com', 'Rakesh');  
$mail → addTo('geeks@zend.com', 'Geek #1');  
$mail → setSubject('DB2Rocks.....');  
$mail → send();  
?>
```

For more information about using Zend_Mail, refer Zend Framework's online manual.

Zend_PDF

The Zend_Pdf module is a PDF (Portable Document Format) manipulation engine written entirely in PHP 5. It can load existing documents, create new documents, modify documents, and save modified documents. Thus, it can help any PHP-driven application dynamically prepare documents in a PDF by modifying an existing template or generating a document from scratch. The Zend_Pdf module supports the following features:

- ▶ Create new document or load existing one
- ▶ Retrieve specified revision of the document
- ▶ Manipulate pages within document
- ▶ Change page order, add new pages, and remove pages from a document
- ▶ Use different drawing primitives (lines, rectangles, polygons, circles, ellipses, and sectors)
- ▶ Allow text drawing using any of the 14 standard (built-in) fonts or your own custom TrueType fonts
- ▶ Handle rotations
- ▶ Perform image drawing
- ▶ Do incremental PDF file update

To see the code example about how to use Zend_PDF, refer to the Zend Framework online manual.

3.4.5 Creating Web services with Zend Framework

Web services are Web-based enterprise applications that use open, XML-based standards and transport protocols to exchange data with calling clients. Many vendors including Amazon, Yahoo, and eBay offer numerous Web services that you can use for free. This section describes the Web services components of

Zend Framework. You can create a simple Web service using `Zend_Service_Flickr` to search a Flickr (a Yahoo company) image.

Zend Framework preview release 0.1.4 comes with these Web services components, which you can use to build a Web services application. We anticipate that more Web service APIs will be added as the Framework matures toward a 1.0 release.

Zend_Service_Amazon

`Zend_Service_Amazon` is a simple API for using Amazon Web services. `Zend_Service_Amazon` has two APIs: A more traditional one that follows Amazon's own API, and a simpler "Query API" for constructing even complex search queries easily. `Zend_Service_Amazon` enables developers to retrieve information appearing throughout Amazon.com Web sites directly through the Amazon Web Services API. Some of the examples are:

- ▶ Store item information, such as images, descriptions, pricing, and more
- ▶ Customer and editorial reviews
- ▶ Similar products and accessories
- ▶ Amazon.com offers
- ▶ ListMania lists

You need to acquire a developer API key to access Amazon Web services. Visit the following Web site to acquire the key:

<http://www.amazon.com/b/102-3922921-6259314?ie=UTF8&node=3435361>

Also, you can refer to the Zend Framework online manual for more information about how to use these Web services.

Zend_Service_Yahoo

`Zend_Service_Yahoo` is a simple API for using many of the Yahoo REST APIs. `Zend_Service_Yahoo` allows you to search Yahoo! Web search, Yahoo! News, Yahoo! Local, Yahoo! Images, and more.

`Zend_Service_Yahoo` enables you to search the Web with Yahoo! using the `webSearch()` method, which accepts a string query parameter and an optional second parameter as an array of search options. For full details and an option list, visit the Yahoo! Web Search Documentation at:

<http://developer.yahoo.com/search/web/V1/webSearch.html>

You can search for Images with Yahoo using Zend_Service_Yahoo's `imageSearch()` method. This method accepts a string query parameter and an optional array of search options. similar to the `webSearch()` method. For full details and an option list, visit the Yahoo! Image Search Documentation at:

<http://developer.yahoo.com/search/image/V1/imageSearch.html>

In order to use the Yahoo Web services API, you must acquire a Web services application ID. To acquire the application ID, submit the form at this Web site:

http://api.search.yahoo.com/webservices/register_application

Zend_Service_Flickr

Flickr is a Yahoo company that offers free services for managing and sharing pictures. Flickr also provides Web services APIs for developers to build Web applications using these APIs.

Zend_Service_Flickr is a simple API for using the Flickr REST Web Service. In order to use the Flickr Web services, you must have an API key. To obtain a key and for more information about the Flickr REST Web Service, visit the Flickr API Documentation at:

<http://www.flickr.com/services/api/>

In order to show you the simplicity of the Web service API, we chose to create a Web service application that searches for images for a user given keyword at Flickr Web site and brings the links to the images. The user can click on the links and display the images. In the Zend Framework MVC architecture, you ideally create a separate controller and templates to build this Web service code, but for simplicity, we created a `photoAction()` method in the existing `IndexController.php`. The code for `photoAction()` is shown in Example 3-40.

Example 3-40 photoAction() method

```
public function photoAction()
{
    //create the Flickr service object
    $flickr = new Zend_Service_Flickr('YOUR_API_KEY');
    $view = Zend::registry('view');
    echo $view → render('flickr.php');
    $keyword = $_POST['keyword'];
    $view → results = $flickr → tagSearch($keyword);
    foreach ($view → results as $entry)
    {
        echo "<br> . $entry → title . <br>";
        $imageId = $entry → id;
        $view → image = $flickr → getImageDetails($imageId);
    }
}
```

```
        foreach ($view → image as $curr)
        {
            echo "<a href=\"\$curr → clickUri\">Click for Image</a>\n";
        }
    }
}
```

As you can see in Example 3-40 on page 142, the first step is to create a Flickr Web service object. In order to do this, you should have your Flickr Web services key ready. For non-commercial use, you obtain this key easily from the Web site just mentioned in the previous section. We render a page for the user to enter a keyword. Each image on Flickr Web site is tagged with a keyword or title. We use a `tagSearch()` function, which takes the keyword as an argument and returns the resultset having the keyword in the tag name. You can also use the `userSearch()` function to return result image information by `userId` or `userEmail`. The following classes are all returned by `tagSearch()` and `userSearch()`:

- ▶ `Zend_Service_Flickr_ResultSet`
- ▶ `Zend_Service_Flickr_Result`
- ▶ `Zend_Service_Flickr_Image`

Once you have a resultset available, you can loop through it and access its properties. In Example 3-40 on page 142, we access the ID and title of an image. We then use the `getImageDetails()` function to access the image location (URI) by its ID. The same Web page shows the image URL with links to different presentations of the same images. Example 3-41 shows the view `flickr.php`.

Example 3-41 Template for displaying flickr Web services output images

```
<html><head>
    <title>Zend Framework Web Services example</title>
</head><body>
<h1>Search for the Flickr Image using Zend Framework web services</h1>
<form action="/index/photo" method="POST">
Type your search keyword: <input type="text" name="keyword"><P>
<INPUT type="submit" value="Submit your search keyword">
</form></body></html>
```

Finally, the Web page (Figure 3-9) accepts the keyword search from the user and gets the images through Flickr Web services and displays them. Similar to the controller design, the URL to access the Web site is:

<http://localhost/index/photo>

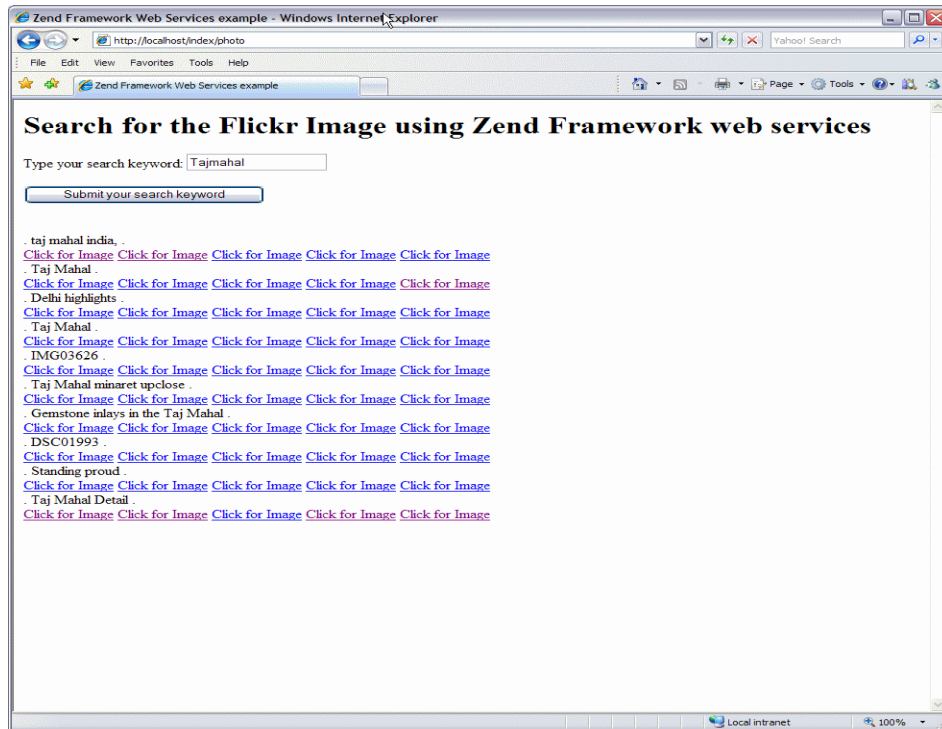


Figure 3-9 Search for Flickr image

3.5 Conclusion

PHP allows rapid Web application development and DB2 9 allows easy manipulation and storage of XML data, plus the capability to perform traditional relational database operations as well.

There are simply far too many interesting PHP application development topics to discuss in this chapter. We hope that the various references and working application examples provided in this chapter have given you some ideas about using the framework with DB2 Express-C to develop your next database driven Web application.

As the volume of XML data generated and used continues to grow in the years to come, easy interoperability between disparate systems seems closer than ever. The XCS will be available to enable developers of all skill levels to quickly and easily create the applications that will allow these systems to talk the common language of XML. An IBM DB2 9 implementation of the XCS further combines powerful XML storage and XQuery technologies with a world-class relational database management system.

Be sure to consult “Related publications” on page 321 for additional reading material, and also be sure to download and try the application code discussed in this chapter.



Application development with C/C++

In this chapter, we discuss:

- ▶ How to set up the environment to build a C/C++ application
- ▶ The fundamentals of using embedded SQL
- ▶ How to set up the environment to build a CLI application
- ▶ The fundamentals of using the Call Level Interface (CLI)
- ▶ Using XML

4.1 Overview

Applications written in the C/C++ programming language can access and manipulate data in a DB2 database using embedded SQL or through the DB2 Call Level Interface (CLI).

Embedded SQL statements in an application can either be of static or dynamic type. If the complete SQL statement and all database objects accessed within the statement are known prior to compile time, then it can be written as a static SQL statement. Otherwise, if the complete SQL statement or any database objects accessed within the statement are not known until runtime, then it must be written as a dynamic SQL statement. Using embedded SQL in an application requires a precompiler to create its associated package in the database. A package is a database object and is needed for processing because it holds the optimized form of an SQL statement.

The DB2 Call Level Interface is an SQL interface that C/C++ applications can use to interact with DB2. The interface follows the ISO CLI and Microsoft's Open Database Connectivity (ODBC) standard. In addition, it provides some useful DB2 Application Programming Interfaces (APIs). Writing C/C++ code to make use of the CLI interface is an alternative to using embedded dynamic SQL.

Whether a C/C++ application uses embedded static SQL, embedded dynamic SQL, or the CLI interface is dependent on several factors. If database statistics do not change much, programs can run faster using embedded static SQL because source files are precompiled and SQL statements do not need to be prepared at runtime. However, if database statistics change often, programs can run faster using embedded dynamic SQL because it uses current database statistics during runtime. If the application wants to make use of dynamic SQL processing, or the application will run against different database products, then using the CLI interface to access DB2 can be the best approach. This is because you do not need to precompile CLI applications, and, hence, there are no application bind files that need to be bound to the database. Note that it is possible for an application to use both embedded static SQL and the CLI interface (which runs dynamic SQL) by writing a CLI application with static SQL modules. You need to take all these factors into consideration when coding a C/C++ application to interact with DB2.

4.1.1 C/C++ development environment setup

Developing C/C++ applications with the DB2 product family requires the DB2 Application Development Client (which is part of the DB2 Client in V9) and a supported C/C++ compiler.

Supported compilers

DB2 Express-C is currently supported on Linux and Windows platforms. At the time of writing, the following C/C++ compilers are supported.

Table 4-1 Supported C/C++ Compiler versions

Platform	Language	Supported compiler versions
Linux	C	GNU/Linux gcc Versions 2.95.3 and 2.96
Linux	C++	GNU/Linux g++ Versions 2.95.3 and 2.96
Windows	C/C++	Microsoft Visual C++® Version 6.0 Microsoft Visual C++ .NET Intel® C++ Compiler for 32-bit applications Version 6 or later

For the latest information about the supported C/C++ compiler versions, refer to the DB2 Application Development Web site at:

<http://www.ibm.com/software/data/db2/udb/ad/>

Setting up the C/C++ environment

To set up the C/C++ environment, follow these steps:

1. Check C/C++ compiler

Ensure a supported C/C++ compiler is installed on a DB2 Express-C supported platform. Check Table 4-1 for a list of supported compiler versions. Refer to the compiler documentation for installation and usage instructions.

2. Check DB2 installation

Ensure the development machine has either DB2 Express-C installed (if the database will be accessed locally) or at least the DB2 Application Development Client (if accessing the database remotely). This ensures you have the necessary precompilers and static libraries to develop C/C++ programs. If a DB2 client is installed, ensure that the client machine can connect successfully to the remote server.

3. Check Windows environment (if applicable)

On Windows development machines, ensure that the INCLUDE environment variable contains %DB2PATH%\INCLUDE as the first directory ahead of any Microsoft Platform SDK include directories. If this is not the case, you can do the following:

- a. Modify the INCLUDE variable at a command prompt by running the command: **set INCLUDE=%DB2PATH%\include;%INCLUDE%**

- b. For development using Visual C++, modify the INCLUDE environment variable in the vcvars32.bat file. Move the %INCLUDE% portion to the beginning of the list of Visual C++ directories.

4.2 Building a C/C++ application using embedded SQL

The steps involved in building a C/C++ application with embedded SQL are:

1. Create C/C++ source files containing programs with embedded SQL statements. The extension of a source file is dependent on the platform and host language (C or C++). The possible file extensions are discussed in 4.3.2, "Precompiler source file extensions" on page 153.
2. Connect to the database, and precompile each source file using the DB2 PREP or PRECOMPILE command.

Depending on the file extension, either the C or C++ precompiler is invoked. The precompiler generates modified source files containing C/C++ language calls for the SQL statements. Depending on the options specified, packages corresponding to the embedded SQL statements in the source files can be created in the database, or bind files containing information about how to create the packages can be produced.

3. Compile the modified source files (and other files without SQL statements) using the C compiler.
4. Using the object files created from the compilation, link the files with DB2 UDB and C libraries to produce an executable program.
5. If packages were not created at precompile time (or if a different database is to be accessed) and bind files were created, the bind files are bound against the database.
6. Run the application.

For our example, assume we finished our C inventory program named `inventory.sqc` on a Windows system using Visual C++. To precompile the program, we can run the following statement shown in Example 4-1.

Example 4-1 Output from running the PREP command

```
db2 prep inventory.sqc
```

```
LINE      MESSAGES FOR inventory.sqc
```

```
-----  
SQL0060W  The "C" precompiler is in progress.  
SQL0091W  Precompilation or binding was ended with "0"  
          errors and "0" warnings.
```

This will produce a modified source file named `inventory.c` and a package in the database.

Alternatively, we can create a bind file and later bind the file to manually create a package. This is known as *deferred binding*. We can accomplish this by using the `BINDFILE` option in the `PRECOMPILE` command.

Example 4-2 Basic statements to create a bind file and package

```
db2 prep inventory.sqc BINDFILE
db2 bind inventory.bnd
```

After precompiling, we need to compile the modified source file. We can do this on the Windows system by running the command:

```
c1 -Zi -Od -c -W2 inventory.c
```

This produces an object file named `inventory.obj`. We can then create an executable file named `inventory.exe` by running the command:

```
link -debug -out:inventory.exe inventory.obj db2api.lib
```

Depending on your compiler and the options you set, the commands for compiling and linking will be different.

4.2.1 Host variables and parameter markers

In a static SQL statement, the complete SQL statement along with the type and length of the data requested is known at precompile time. The only missing information is the actual data values for the SQL statement. Acting as placeholders for the missing data, host variables need to be defined by the application to represent the remaining information in the SQL statement. Host variables are represented by prefixing a colon (`:`) in front of the variable name.

For a dynamic SQL statement, parameter markers are represented as a question mark (`?`) for any variable information in an SQL statement.

4.3 A simple C inventory program using embedded SQL

To illustrate the basic fundamentals of building a C/C++ application, let us develop a simple C inventory program for a company to store inventory information about its products, and the associated quantity and location of each product.

For simplicity, we work with a single table to manage the inventory for the company. You need to create the SAMPLE database provided with DB2 Express-C for this program, because it will interact with the INVENTORY table. If the database does not exist, refer to Chapter 1, “DB2 application development overview” on page 1 for steps about how to create the database.

The application interacts with the user to perform the following operations:

- ▶ Enter a new product:
 - New products need to be entered into the database. For a new product, its product ID/barcode, quantity on hand, and location information need to be added to the INVENTORY table.
 - An error will occur if the product already exists in the database.
- ▶ Update quantity/Update location:
 - Any changes in the quantity or location of a product need to be reflected in the table.
 - An error will occur if the product does not exist in the database.
- ▶ Get information about a part:
 - The user can query the database for a product to get information about the quantity at hand or its location.
 - An error will occur if the product does not exist in the database.

4.3.1 The INVENTORY table

The INVENTORY table in the SAMPLE database consists of three columns named PID, QUANTITY, and LOCATION. They store the ID/barcode, quantity on hand, and the location of a product. The PID column is the unique identifier for a product and must be distinct within the table. By default, the INVENTORY table in the SAMPLE database is populated with the records shown in Example 4-3.

Example 4-3 Sample data from the INVENTORY table

PID	QUANTITY	LOCATION
100-100-01	5	-
100-101-01	25	Store
100-103-01	55	Store
100-201-01	99	Warehouse

4.3.2 Precompiler source file extensions

To start off, we need to give the C program the appropriate file extension, which the precompiler can recognize. C/C++ precompilers are needed to convert embedded SQL statements in each C/C++ source file to DB2 Runtime API calls to the DB2 engine.

Unless otherwise specified by the BINDFILE option, a package is generated in the target database during precompile time and is needed for SQL to be executed. A *package* is a database object containing sections, which correspond to embedded SQL statements in a source program module. A section contains a compiled/optimized SQL statement.

The precompiler can optionally defer the creation of a package and create a bind file, which contains all the information needed to create a package. To create a package, the bind file will need to later be manually bound to the database. This is known as *deferred binding*. Deferred binding provides more flexibility because applications only need to be precompiled once (without a package being created), but can be bound against multiple databases later to create a package using the one bind file. This means the same application can access several databases, instead of just the one database it was precompiled against.

Once a package is created, optimizer access plans are created for static embedded SQL statements from the current database statistics during bind time. Dynamic embedded SQL statements will get an access plan created during runtime.

Table 4-2 lists the C/C++ source file extensions required by the precompiler, along with the file extensions given to the modified source output files.

Table 4-2 Precompiler file extensions

	Source file extension (input)	Modified source file extension (output)
C (Windows/Linux)	.sqc	.c
C++ (Linux)	.sqC	.C
C++ (Windows)	.sqx	.cxx

If we name our example C program `inventory.sqc`, and precompile the program once it is finished, this produces a modified source file with the file extension of `.c`. The OUTPUT option can be specified in the PREP/PRECOMPILE command to override the name of the modified source file.

4.3.3 Inventory program code template

We start off with a simple code skeleton of the application (see Example 4-5 on page 155). This C program prompts the user for an inventory operation and calls the appropriate functions to insert (`add_product()`), update (`update_product()`), or perform queries (`query_update()`) against the `INVENTORY` table in the `SAMPLE` database. Applications can be written to contain both static and dynamic embedded SQL statements. For demonstration purposes, this application contains both types.

Example 4-4 Code template of inventory.sqc

```
/* Source File Name: inventory.sqc
**
** This simple C program will manage the inventory of
** products for a company.
**
*****/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void add_product(void);
void update_product(void);
void query_product(void);

main()
{
    int option;

    while(1){
        printf("-----\n");
        printf(" SIMPLE INVENTORY MANAGEMENT SYSTEM \n\n");
        printf(" 1. Add New Product                \n");
        printf(" 2. Update Product Info (Quantity or Location) \n");
        printf(" 3. Query Product                      \n");
        printf(" 4. Exit                               \n\n");
        printf(" Enter option: ");
        scanf("%d", &option);

        switch(option) {
            case 1:add_product();
                break;
            case 2:update_product();
        }
    }
}
```

```

        break;
    case 3:query_product();
        break;
    case 4:return 0;
    default : printf(" Error: Option not recognized\n");
        break;
    }

    printf ("\n");
}

}

void add_product(void){...}
void update_product(void){...}
void query_product(void){...}

```

Dynamic embedded SQL is used to update information about a product. The user can choose to update the quantity or location of a particular product. The application will prompt the user to specify which piece of information about a product they wish to update. The complete SQL statement will not be known at precompile time because the column of the table (either QUANTITY or LOCATION) to be updated is not known. The column of the table to be updated will only be known at runtime once the user supplies the data. Only at that time can the SQL statement be built and submitted to the database for processing.

Static embedded SQL is used in adding a new product and querying for information about a product, because the complete SQL statement will exist at precompile time.

The system will provide a menu of basic inventory operations to the user, and look like Example 4-5.

Example 4-5 Menu of inventory operations

```

-----
SIMPLE INVENTORY MANAGEMENT SYSTEM

1. Add New Product
2. Update Product Info (Quantity or Location)
3. Query Product
4. Exit

Enter option:

```

4.3.4 Host variable declarations

Host variables used in static SQL statements are declared in a DECLARE SECTION. For example, let us declare the static host variables needed for our program, see Example 4-6.

Example 4-6 Host variable declarations

```
EXEC SQL BEGIN DECLARE SECTION;
    char PID[11];
    sqlint32 quantity;
    char location[129];
EXEC SQL END DECLARE SECTION;
```

The EXEC SQL portion of the declaration is needed to indicate the beginning of an embedded SQL statement and must be terminated by a semicolon. The normal form is:

```
EXEC SQL <Standard SQL statement>;
```

In the INVENTORY table in the SAMPLE database, the PID and LOCATION columns are of type VARCHAR(10) and VARCHAR(128), respectively. In the above declaration, we have defined the size of those columns to be of one more character to take into account the null terminator. This will ensure that returned data values from the database do not get truncated. It is permissible to have more than one DECLARE section in a source file, although all host variables declared must be distinct within the source file.

After declaring the host variables, we have the option of using them as input or output variables in an SQL statement. Example 4-7 shows the permissible usage of host variables within an SQL statement.

Example 4-7 SQL Statement with both input and output host variables

```
SELECT quantity, location into :quantity, :location from inventory
where PID=:PID;
```

The input host variable for the above statement is PID and a value for it will be supplied to the database by the application. The output host variables are QUANTITY and LOCATION, and its data values will be returned from the database to the application. In both cases, a colon (:) must precede the host variable name.

4.3.5 Using db2bfd to display host variable declarations

If a bind file is created during precompile time, we can use a DB2 bind file description tool named `db2bfd` to look at the contents of the bind file. This may be useful for debugging and can display information such as SQL statements or host variables declared. Table 4-3 lists the `db2bfd` command options.

Table 4-3 *db2bfd* command options

db2bfd option	Information
-s	SQL statements
-b	Bind file header
-v	Host variables declared
-h	Help

For example, let us precompile the source file and add the option to create a bind file by running the command:

```
db2 prep inventory.sqc bindfile
```

This will produce a bind file named `inventory.bnd`.

Running the command `db2bfd -v inventory.bnd` provides CLP output as shown in Example 4-8.

Example 4-8 *db2bfd* sample output

```
inventory.bnd: Host Variables = 3
```

Type	SQL Data Type	Length	Alias	Name_Len	Name	UDT Name
460	C STRING	11	H00001	3	PID	
496	INTEGER	4	H00002	8	quantity	
460	C STRING	128	H00003	8	location	

4.3.6 Using db2dclgn to generate host variable declarations

DB2 can simplify development by automatically generating host variable declarations for a specified database table.

In Example 4-9 on page 158, the command will generate an `.h` output file with the same name as the table. It will contain the host variable declarations.

```
db2dc1gn -d sample -t inventory
```

```
inventory.h:
```

```
struct
{
    struct
    {
        short length;
        char data[10];
    } pid;
    sqlint32 quantity;
    struct
    {
        short length;
        char data[128];
    } location;
} inventory;
```

4.3.7 Connecting to a database

Let us connect to the database within the application and start implementing the operations in our application. To connect to the database, we use the `CONNECT TO` statement. We add the following SQL statement into the main function of our program.

```
EXEC SQL CONNECT TO sample;
```

Alternatively, we can choose to connect to the database using a particular user ID and password:

```
EXEC SQL CONNECT TO sample USER :uid USING :pwd;
```

If this method is chosen, host variables for the user ID and password need to be declared for use in the `CONNECT` statement. In our simple inventory application, we just connect to the `SAMPLE` database using the default user ID.

4.3.8 Disconnecting from a database

To disconnect from the database, we can add the following statement at the end of our program.

```
EXEC SQL CONNECT RESET;
```

4.3.9 The SQL Communications Area (SQLCA)

The *SQL Communications Area* (SQLCA) is a DB2 data structure useful in obtaining information pertaining to the processing of an SQL statement or an API call.

As the SQLCA is updated by the database manager after each SQL statement, it is important we examine the structure to ensure that any unexpected errors will be handled by the application.

The SQLCA structure provides important information such as:

▶ **SQLCODE:**

This is an integer return code indicating whether the most recent SQL statement processed was successful or not. The value is 0 if successful, 100 if a query yields no results, positive if a warning occurred, and negative if the processing was unsuccessful and an error was returned.

▶ **SQLSTATE:**

This is a five digit character length field that contains a return code from the last SQL statement processed. The return code is consistent with the SQL Standard.

▶ **SQLERRD:**

This is an integer array containing six pieces of information related to the last SQL statement processed that can be useful when an error occurs.

To define the SQLCA, we can add the following statement to our program:

```
EXEC SQL INCLUDE SQLCA;
```

Alternatively, we can include the SQLCA header file and use it to declare an application variable:

```
#include <sqlca.h>
struct sqlca mySqlca;
```

If we had additional source files, we can add the following lines to the other source files:

```
#include <sqlca.h>
extern struct sqlca sqlca
```

4.3.10 Quick SQLCA example

Let us see a quick example of how we can use the SQLCA to get information related to the processing of an SQL statement. Let us connect to the database with an incorrect database name using the following statement:

```
EXEC SQL CONNECT TO sample;
```

Let us also add the following lines in the code to check whether the database connection was successful using the SQLCA:

```
if (sqlca.sqlcode!=0){
    printf("Database connection error occurred.
    Sqlcode=%d\n",sqlca.sqlcode);
    return;
}
```

We then precompile, compile, link, and run the program. The following is returned by the application:

```
Database connection error occurred.  Sqlcode=-1013
```

For debugging and illustrative purposes, we can check out the fields of the SQLCA to see what information is provided. Normally, not all SQLCA fields are needed to diagnose an application problem. We print the fields of the SQLCA as shown in Example 4-10.

Example 4-10 Code to print the SQLCA fields

```
printf("sqlcabc: %d\n", sqlca.sqlcabc); // Length of SQLCA
printf("sqlcode: %d\n", sqlca.sqlcode); // SQL Return Code
printf("sqlerrml: %d\n", sqlca.sqlerrml); // Length of sqlerrmc
printf("sqlerrmc: %s\n", sqlca.sqlerrmc); // Message tokens
printf("sqlerrd[0]: %d\n", sqlca.sqlerrd[0]);
printf("sqlerrd[1]: %d\n", sqlca.sqlerrd[1]);
printf("sqlerrd[2]: %d\n", sqlca.sqlerrd[2]);
printf("sqlerrd[3]: %d\n", sqlca.sqlerrd[3]);
printf("sqlerrd[4]: %d\n", sqlca.sqlerrd[4]);
printf("sqlerrd[5]: %d\n", sqlca.sqlerrd[5]);
printf("sqlwarn: %s\n", sqlca.sqlwarn); // Warning indicators
printf("sqlstate: %s\n", sqlca.sqlstate); // Return Code
```

This produces the output shown in Example 4-11 on page 161.

Example 4-11 Sample output of values stored in the SQLCA

```
sqlcaid: SQLCA   ē
sqlcabc: 136
sqlcode: -1013
sqlerrml: 7
sqlerrmc: SAMPSE
sqlerrd[0]: 0
sqlerrd[1]: 0
sqlerrd[2]: 0
sqlerrd[3]: 0
sqlerrd[4]: 0
sqlerrd[5]: 0
sqlwarn:                42705
sqlstate: 42705
```

Looking up the SQLCODE and SQLSTATE, we discover that the problem resulted because the database manager could not find the database specified in the application.

Alternatively, we can use a DB2 API called `sqlaintp` defined in `sql.h` to help get more information about the error. This function will return the error text for an SQLCODE in the SQLCA. An example of using this function is shown in Example 4-12.

Example 4-12 Using the `sqlaintp` API to get error info

```
char errText[1000];
...
sqlaintp(errText,999,0,&sqlca);
printf("%s\n",errText);
```

The function call above retrieves the SQLCODE error message and stores it in a buffer named `errText`. As input to the function, we provide the string buffer, which stores the error message (`errText`), the size of the string buffer (999), the maximum width of each line of the message text ('0' means no line breaks needed in the text), and the pointer to the SQLCA structure (`sqlca`).

The function call will generate the following message:

```
SQL1013N The database alias name or database name "SAMPSE" could not
be found.  SQLSTATE=42705
```

After all this debugging, it appears the name of the database specified in our code was incorrect. The line needs to be corrected to:

```
EXEC SQL CONNECT TO sample;
```

4.3.11 Inserting data

The users of the inventory system in our example need to be able to add a new product into the database. The user might have the session shown in Example 4-13.

Example 4-13 Sample session adding a product into inventory program

```
-----  
SIMPLE INVENTORY MANAGEMENT SYSTEM  
  
1. Add New Product  
2. Update Product Info (Quantity or Location)  
3. Query Product  
4. Exit  
  
Enter option: 1  
-----  
Add A Product  
  
Enter Product ID : 101-101-10  
Enter Quantity   : 50  
Enter Location   : Warehouse  
  
Product successfully added.  
-----
```

When the user chooses the option to add a new product, this will invoke the `add_product()` function within the application. We need to write an `INSERT` statement to add a record to the `INVENTORY` table using the values supplied by the user.

The complete definition of the `add_product()` function is shown in Example 4-14.

Example 4-14 Complete definition of the `add_product()` function

```
void add_product(void)  
{  
    printf("-----\n");  
    printf(" Add A Product \n\n");  
    printf(" Enter Product ID : ");  
    scanf("%s", &PID);  
    printf(" Enter Quantity   : ");  
    scanf("%d", &quantity);  
    printf(" Enter Location   : ");  
    scanf("%s", &location);  
}
```

```

EXEC SQL INSERT INTO inventory (PID, quantity, location) VALUES
(:PID, :quantity, :location);

if (sqlca.sqlcode!=0) {
    if (sqlca.sqlcode == -803)
        printf("\n Error: Product already exists in the database.\n");
    else
        printf("\n Error: Problem occurred when adding a new product.
Sqlcode: %d Sqlstate: %s\n",sqlca.sqlcode, sqlca.sqlstate);
    return;
}

EXEC SQL COMMIT;
printf("\n Product successfully added.");
}

```

In Example 4-14 on page 162, we prompt the user for information on the product to be added and populate the host variables with the supplied values. We then use the host variables in the INSERT statement and check whether the processing was successful.

Note that we add a COMMIT after the execution of the INSERT. This is because when a database connection is initially established, the application starts a transaction with any executable SQL statement such as SELECT, INSERT, CREATE, GRANT, and so on. Within the transaction, any number of SQL statements can be issued. This transaction is considered to be an *atomic* unit of work where either all or none of the changes within a transaction are made. To end the transaction, a COMMIT or a ROLLBACK must be issued. Issuing a COMMIT will apply all the changes to the database, whereas issuing a ROLLBACK cancels any changes to be made to the database.

4.3.12 Retrieving data

There are multiple ways to retrieve data from a database. Depending on whether a single row or multiple row result set is returned, this determines whether a cursor will be used. A *cursor* is a mechanism that is used to process each row of a result set.

For example, assume we want to find all the products available at the store in our INVENTORY table. Because it is possible that more than one row is returned from the query, we use a cursor to retrieve and process the rows. Example 4-15 on page 164 shows how to use a cursor.

Example 4-15 Using a cursor

```
EXEC SQL DECLARE c1 CURSOR FOR SELECT PID from inventory where
location='Store';

EXEC SQL OPEN c1;
EXEC SQL FETCH c1 INTO :PID;
printf("%s\n",PID);
...
EXEC SQL CLOSE c1;
EXEC SQL COMMIT;
```

In Example 4-15, a cursor named c1 is declared to hold a resultset of product IDs. To use the cursor, the cursor is opened and data is retrieved into the PID host variable. Once we are finished processing the resultset, the cursor should be closed and a COMMIT executed to release any resources held.

In the inventory program we are developing, we need to retrieve information about a single product. Because products are unique within the INVENTORY table, we will only be retrieving one row back from the database (if the product exists). This means that instead of a cursor, we can use a SELECT INTO statement which will directly populate the host variables. A sample session from querying a product is shown in Example 4-16. The complete definition of the query_product() function is shown in Example 4-17 on page 165.

Example 4-16 Sample session querying a product in the inventory program

```
-----
SIMPLE INVENTORY MANAGEMENT SYSTEM

1. Add New Product
2. Update Product Info (Quantity or Location)
3. Query Product
4. Exit

Enter option: 3
-----
Query A Product

Enter Product ID : 101-101-10
Found Quantity   : 50
      Location    : Warehouse

-----
```

Example 4-17 Complete definition of the query_product() function

```
void query_product(void)
{
    printf("-----\n");
    printf(" Query A Product \n\n");
    printf(" Enter Product ID : ");
    scanf("%s", &PID);

    EXEC SQL WHENEVER SQLERROR GOTO errorFound;
    EXEC SQL WHENEVER NOT FOUND GOTO PIDNotFound;

    EXEC SQL SELECT quantity, location INTO :quantity, :location from
inventory where PID=:PID;

    printf(" Found Quantity   : %d\n",quantity);
    printf("           Location   : %s\n",location);
    EXEC SQL COMMIT;

    return;

    errorFound: {
        printf("\n Error: Problem occurred.  Sqlcode: %d Sqlstate:
%s\n",sqlca.sqlcode, sqlca.sqlstate);
        return;
    }
    PIDNotFound: {
        printf("\n Error: Product ID was not found. \n");
        return;
    }
}
```

4.3.13 Indicator variables

When a value is nullable, an application needs to check whether the value of a column returned is null or not. To do this using embedded SQL, we can use indicator variables. For example, if we want to check whether the quantity and location data value returned from the database is null, we can modify the statement in Example 4-17 to the one shown in Example 4-18 on page 166.

Example 4-18 Using indicator values to check nullability

```
EXEC SQL SELECT quantity, location INTO :quantity INDICATOR :ind1,  
:location INDICATOR :ind2 from inventory where PID=:PID;
```

The INDICATOR keyword and the indicator host variable are added after each host variable to be checked. We need to ensure that the indicator host variables ind1 and ind2 are declared in the DECLARE section of the program. As the indicator variable is of SMALLINT data type, the C data type can be declared to be short. An indicator host variable with a value of -1 is null, any other value is considered not-NULL. The INDICATOR keyword is optional. Example 4-19 shows the same SQL statement without the INDICATOR keyword.

Example 4-19 Using indicator without INDICATOR keyword

```
EXEC SQL SELECT quantity, location INTO :quantity :ind1, :location :ind2 FROM  
inventory where PID=:PID;
```

Note: There is no comma between a host variable and its indicator.

4.3.14 The WHENEVER Statement

Within an application, we can specify how to handle certain SQL conditions such as if an SQL error, warning, or a not found condition (SQLCODE of +100) is returned from the database.

There are three main types of the WHENEVER clause that can be in an embedded SQL statement:

```
EXEC SQL WHENEVER NOT FOUND <action>  
EXEC SQL WHENEVER SQLERROR <action>  
EXEC SQL WHENEVER SQLWARNING <action>
```

The possible actions are CONTINUE, GOTO <label>, or GO TO <label>.

In Example 4-17 on page 165, we added WHENEVER clauses into our query_product() function. If a NOT FOUND condition occurs, we will print out the appropriate error message. Otherwise, we print a more generic error message.

4.3.15 Preparing SQL statements

We have been working with embedded static SQL statements in the add_product() and query_product() functions in our sample inventory program. However, for the inventory operation where a column is to be updated, we need to use embedded dynamic SQL and dynamically prepare the SQL statement.

This will finally generate an executable access plan in the package. Once that is done, we can execute the prepared SQL statement or declare a cursor to handle the result set.

For example, let us say we wanted to delete a record from the INVENTORY table. Once we prepare the SQL statement, we can execute the statement multiple times using different parameter marker values but the same application package.

Example 4-20 Preparing a DELETE statement

```
strncpy(deleteStmt,"delete from inventory where PID=?",
sizeof(deleteStmt));
EXEC SQL PREPARE stmt FROM :deleteStmt;
strncpy(value," 100-100-01",sizeof(value));
EXEC SQL EXECUTE stmt USING :value;
strncpy(value," 100-101-01",sizeof(value));
EXEC SQL EXECUTE stmt USING :value;
EXEC SQL COMMIT;
```

Assuming we have declared the deleteStmt and value host variables in a DECLARE section, the code fragment in Example 4-20 prepares a statement with a single parameter marker. The prepared statement then gets executed multiple times using different data values. It is important to remember that a COMMIT must be issued in order for the changes to the database to occur.

You might notice that preparing the DELETE statement in Example 4-20 is quite unnecessary, because it can be done using static embedded SQL and host variables since we know the complete SQL statement. It can be rewritten as Example 4-21.

Example 4-21 Rewriting Example 4-22 as a static SQL statement

```
strncpy(value," 100-100-01",sizeof(value));
EXEC SQL DELETE FROM INVENTORY WHERE PID=:value;
strncpy(value," 100-101-01",sizeof(value));
EXEC SQL DELETE FROM INVENTORY WHERE PID=:value;
EXEC SQL COMMIT;
```

If a statement can be written as static SQL, you might want to consider avoiding preparing a statement dynamically using parameter maker to avoid run-time costs incurred with a PREPARE. However, if static SQL is run on databases, which are continuously changing (which means the statistics keep changing), dynamic SQL might be more suitable. Also, dynamic SQL offers more flexibility than static SQL has to offer. In your application development, you need to keep these things in mind and choose your approaches wisely.

If an SQL statement does not contain host variables, nor parameter markers, and it does not return a result set, then the EXECUTE IMMEDIATE statement can be used. This statement prepares and executes a statement in one step. For example, in our inventory program, we need to update information about a product in the INVENTORY table as specified by the user. Because the column to be updated is unknown prior to precompile time, dynamic SQL must be used and we need to prepare the statement. Once we get the column to be updated and its value from the user, we can build the SQL string and then use the EXECUTE IMMEDIATE statement to prepare and execute in one step. This is shown in Example 4-22. A sample user session is shown in Example 4-23 on page 169. The completed update_product() function can be found in the complete program definition shown in Example 4-24 on page 169.

Example 4-22 Updating a column using EXECUTE IMMEDIATE

```
if (option==UPDATE_LOCATION) // Ensure single quotes enclose the
location value
    sprintf(updateStmt,"update inventory set %s='%s' where PID='%s'",
updateField, updateValue, PID);
else
    sprintf(updateStmt,"update inventory set %s=%s where PID='%s'",
updateField, updateValue, PID);

EXEC SQL EXECUTE IMMEDIATE :updateStmt;
EXEC SQL COMMIT;
```

Example 4-23 Sample session updating a product in the inventory program

```
-----  
SIMPLE INVENTORY MANAGEMENT SYSTEM  
  
1. Add New Product  
2. Update Product Info (Quantity or Location)  
3. Query Product  
4. Exit  
  
Enter option: 2  
-----  
Update A Product  
  
1. Update quantity  
2. Update location  
  
Enter option: 1  
  
Enter product ID to update: 101-101-10  
Enter new quantity : 20  
  
Product successfully updated.  
-----  
-----
```

4.3.16 Complete C inventory program

After adding the data retrieve and manipulation code into the inventory program template shown in Example 4-4 on page 154, we have a complete C inventory program as shown in Example 4-24.

Example 4-24 inventory.sqc

```
/*  
*****  
** Source File Name: inventory.sqc  
**  
** This simple C program will manage the inventory of  
** products for a company  
**  
*****/  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
  
EXEC SQL BEGIN DECLARE SECTION;
```

```

    char PID[11];
    sqlint32 quantity;
    char location[129];
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;

void add_product(void);
void update_product(void);
void query_product(void);

main()
{
    int option;

    // Connect to the SAMPLE database
EXEC SQL CONNECT TO sample;
    if (sqlca.sqlcode!=0){
        printf("Database connection error occurred.  Sqlcode: %d
Sqlstate: %s\n",sqlca.sqlcode, sqlca.sqlstate);
        return;
    }

    while(1){
        printf("-----\n");
        printf(" SIMPLE INVENTORY MANAGEMENT SYSTEM \n\n");
        printf(" 1. Add New Product                \n");
        printf(" 2. Update Product Info (Quantity or Location) \n");
        printf(" 3. Query Product                            \n");
        printf(" 4. Exit                                     \n\n");
        printf(" Enter option: ");
        scanf("%d", &option);

        switch(option) {
            case 1:add_product();
                break;
            case 2:update_product();
                break;
            case 3:query_product();
                break;
            case 4:return 0;
            default : printf(" Error: Option not recognized\n");
                break;
        }
    }
}

```

```

        printf ("\n");
    }
    EXEC SQL CONNECT RESET;

}

/*****
** This function will add a product into the
** INVENTORY table. It demonstrates how to write a
** simple static SQL statement and work with the
** SQLCA to get back information on the processing of
** an SQL statement
*****/
void add_product(void)
{
    printf("-----\n");
    printf(" Add A Product \n\n");
    printf(" Enter Product ID : ");
    scanf("%s", &PID);
    printf(" Enter Quantity   : ");
    scanf("%d", &quantity);
    printf(" Enter Location    : ");
    scanf("%s", &location);

    EXEC SQL INSERT INTO inventory (PID, quantity, location) VALUES
(:PID, :quantity, :location);

    if (sqlca.sqlcode!=0) {
        if (sqlca.sqlcode == -803)
            printf("\n Error: Product already exists in the database.\n");
        else
            printf("\n Error: Problem occurred when adding a new product.
Sqlcode: %d Sqlstate: %s\n",sqlca.sqlcode, sqlca.sqlstate);
        return;
    }

    EXEC SQL COMMIT;
    printf("\n Product successfully added.");
}

/*****
** This function will update information on an existing
** product into the INVENTORY table. It demonstrates
** how to write dynamic SQL, and use the WHENEVER
** statement.
*****/

```

```

*****/
void update_product(void)
{
    char updateField[9];
    char updateValue[128];
    enum {UPDATE_QUANTITY, UPDATE_LOCATION} option;

    EXEC SQL BEGIN DECLARE SECTION;
        char updateStmt[200];
    EXEC SQL END DECLARE SECTION;

    EXEC SQL WHENEVER SQLERROR GOTO errorFound;
    EXEC SQL WHENEVER NOT FOUND GOTO PIDNotFound;

    printf("-----\n");
    printf(" Update A Product \n\n");
    printf(" 1. Update quantity \n");
    printf(" 2. Update location \n\n");

    while(1) {
        printf(" Enter option: ");
        scanf("%d", &option);

        option--; // Enum starts at 0. Our options start at 1.

        if (option==UPDATE_QUANTITY) {
            strncpy(updateField,"quantity",sizeof(updateField));
            break;
        }
        else if (option==UPDATE_LOCATION) {
            strncpy(updateField,"location",sizeof(updateField));
            break;
        }
        else {
            printf(" Error: Option not recognized\n\n");
        }
    }

    printf("\n Enter product ID to update: ");
    scanf("%s", &PID);
    printf(" Enter new %s : ", updateField);
    scanf("%s", &updateValue);

    if (option==UPDATE_LOCATION) // Ensure single quotes enclose the
location value

```

```

        sprintf(updateStmt,"update inventory set %s='%s' where PID='%s'",
updateField, updateValue, PID);
    else
        sprintf(updateStmt,"update inventory set %s=%s where PID='%s'",
updateField, updateValue, PID);

    EXEC SQL EXECUTE IMMEDIATE :updateStmt;
    EXEC SQL COMMIT;

    printf("\n Product successfully updated.");
    return;
    errorFound: {
        printf("\n Error: Problem occurred.  Sqlcode: %d Sqlstate:
%s\n",sqlca.sqlcode, sqlca.sqlstate);
        return;
    }
    PIDNotFound: {
        printf("\n Error: Product ID was not found. \n");
        return;
    }
}

/*****
** This function will query for information on a
** product in the INVENTORY table using static SQL
*****/
void query_product(void)
{
    printf("-----\n");
    printf(" Query A Product \n\n");
    printf(" Enter Product ID : ");
    scanf("%s", &PID);

    EXEC SQL WHENEVER SQLERROR GOTO errorFound;
    EXEC SQL WHENEVER NOT FOUND GOTO PIDNotFound;

    EXEC SQL SELECT quantity, location INTO :quantity, :location from
inventory where PID=:PID;

    printf(" Found Quantity   : %d\n",quantity);
    printf("      Location       : %s\n",location);
    EXEC SQL COMMIT;

    return;
}

```

```

    errorFound: {
        printf("\n Error: Problem occurred.  Sqlcode: %d Sqlstate:
%s\n",sqlca.sqlcode, sqlca.sqlstate);
        return;
    }
    PIDNotFound: {
        printf("\n Error: Product ID was not found. \n");
        return;
    }
}
}

```

4.3.17 The SQL Descriptor Area (SQLDA)

The SQL Descriptor Area (SQLDA) is a structure that can provide information about dynamic SQL statements that are prepared and executed. With DESCRIBE, PREPARE, OPEN, FETCH, CALL, and EXECUTE statements, an application can retrieve information about the prepared statement, or input and output parameters of the SQL statement. In PREPARE and DESCRIBE, an SQLDA provides information to an application program about a prepared statement. In OPEN, EXECUTE, FETCH, and CALL, an SQLDA describes host variables.

The SQLDA is made up of header and SQLVAR describing its structure. Within it, there can be multiple SQLVAR elements defined. An individual SQLVAR structure can hold information regarding:

- ▶ A column of a table in a DESCRIBE or PREPARE statement
- ▶ A variable in a FETCH, EXECUTE, CALL, or OPEN statement

The fields, which describe the SQLDA, contain information on the size of the structure, whether the SQLDA has doubled in size (which will happen if LOB objects are used), the number of SQLVAR elements allocated, and the number of columns/variables in the SQL statement.

Depending on whether the SQLDA is being used to store input or output information of an SQL statement, SQLVAR elements can contain information about data types, lengths, column names, code pages, and much more.

To define the SQLDA, we can add the following statement to our program:

```
EXEC SQL INCLUDE SQLDA;
```

Alternatively, we can include the SQLDA include file and declare an application variable to make use of the structure:

```
#include <sqlda.h>
struct sqlda *mysqlda;
```

Once we define the SQLDA structure, we can use it within SQL statements to store input or output information as needed. For example, we can use it within a prepare statement where *<sqlstatement>* is a variable storing a dynamic SQL statement:

```
EXEC SQL PREPARE stmt INTO :*mySqllda FROM <sqlstatement>;
```

The above statement will prepare and store information into the SQLDA structure about the dynamic SQL statement specified in the statement. This is equivalent to the following two statements:

```
EXEC SQL PREPARE stmt FROM <sqlstatement>;
EXEC DESCRIBE stmt INTO :*mySqllda;
```

Note that the example above only describes the output rows of a resultset or in the case of a stored procedure CALL, the INOUT and OUT arguments. To get information about input parameter markers, you need to add an INPUT INTO :*myInputSQLDA clause.

Let us try an example and see what is stored in the SQLDA. We include the sqlca.h header file, declare the mysqlda variable in our code, and define the variables in Example 4-24.

Example 4-25 Setting up the SQLDA

```
char test[40];
mysqlda=(struct sqlda*) malloc(SQLDASIZE(2));
mysqlda->sqln=2;
```

In Example 4-25, we define a buffer named test to store the SQL string that we will be working with. Also, we allocate an SQLDA structure, which will hold two SQLVAR elements by using the SQLDASIZE macro, and as required, set a field named sqln which stores the number of SQLVAR elements needed. We defined 2 SQLVAR structures to be used for our application because we will be working with at most two table columns.

Next, we assign test to hold our SQL statement. We will be querying for the product ID (PID) field from the INVENTORY table. We prepare the statement, using the SQLDA to hold information about the column being returned, and subsequently print fields of the SQLDA structure.

Example 4-26 Printing SQLDA Fields

```
strncpy (test,"SELECT PID FROM inventory", sizeof(test));

/*****
Call PREPARE INTO to prepare and describe the SQL statement. The
statement is equivalent to :

EXEC SQL PREPARE stmt INTO :*mysqlda FROM :test;
EXEC SQL EXECUTE stmt USING DESCRIPTOR :*mysqlda;
*****/

EXEC SQL PREPARE stmt INTO :*mysqlda FROM :test;

printf("Number of columns(sqld)= %d\n", mysqlda->sqld);
printf("Number of SQLVARs(sqln)= %d\n", mysqlda->sqln);
printf("Column name (sqlname.data)= %s\n",
mysqlda->sqlvar[0].sqlname.data);
printf("Column name length (sqlname.length)= %d\n",
mysqlda->sqlvar[0].sqlname.length);
printf("Column datatype (sqltype)= %d\n", mysqlda->sqlvar[0].sqltype);
printf("Column length (sqlllen)= %d\n", mysqlda->sqlvar[0].sqlllen);
```

Running this produces the following output.

Example 4-27 Sample output of values stored in the SQLDA

```
Number of columns(sqld)= 1
Number of SQLVARs(sqln)= 2
Column name (sqlname.data)= PID
Column name length (sqlname.length)= 3
Column datatype (sqltype)= 448
Column length (sqlllen)= 10
```

The result in Example 4-27 informs us that there are two SQLVAR elements defined (sqln), and one column in the resultset returned (sqld). The name of the column (sqlname.data) is PID which is of length (sqlname.length) 10 and of type (sqltype) SQL_VARCHAR (448).

Although we defined two SQLVAR elements, we only really needed one SQLVAR element because there is only one column returned by the SQL statement. In the PREPARE of a SELECT statement, ideally the number of SQLVAR elements defined should be equal to the number of columns expected from the result set. If more SQLVARs are defined than is needed, more memory

would be unnecessarily used. If fewer SQLVARs are defined, there will not be enough SQLVAR elements to describe all the columns being returned.

4.4 Building a C/C++ application using CLI

In this section, we show you the fundamentals of building a C/C++ application using CLI. We discuss first some basic concepts and the environment setup. We then guide you step-by-step to build an application starting from a simple code template.

4.4.1 CLI handles

A Call Level Interface (CLI) application needs to initially set up data structures and variables to manage work such as connecting to the database, running SQL statements, and disconnecting from the database. Luckily, data objects called *handles* exist to make things easier. Handles can be allocated, used for processing, and then freed. The possible CLI handles are:

- ▶ Environment (SQL_HANDLE_ENV)
This object is the base handle, which holds information and provides a context for all connections. Typically, only a single environment handle exists for an application.
- ▶ Connection (SQL_HANDLE_DBC)
This object holds information about a connection to a single database and provides a context for all statements executed against that database. A CLI application can connect to multiple databases. This means that there can be multiple connection handles associated with a single environment handle.
- ▶ Statement (SQL_HANDLE_STMT)
This object contains information about the processing of an SQL statement. This must be associated with a connection handle.
- ▶ Descriptor (SQL_HANDLE_DESC)
This handle holds information about either parameter markers or columns in a resultset. It can be associated with either a statement or connection handle.

Figure 4-1 on page 178 illustrates the relationship of the handles. Connection handles can only be defined under the context of an environment handle, and statement handles can only be defined under the context of a connection handle. Descriptor handles can be optionally allocated to associate with a connection or statement handle. Handle allocation is done in sequence. The environment handle must be declared first, then the connection handle is declared under the environment handle, and then the statement handle is declared under the

connection handle. When freeing the handles, all child handles should be freed explicitly before freeing the parent handle.

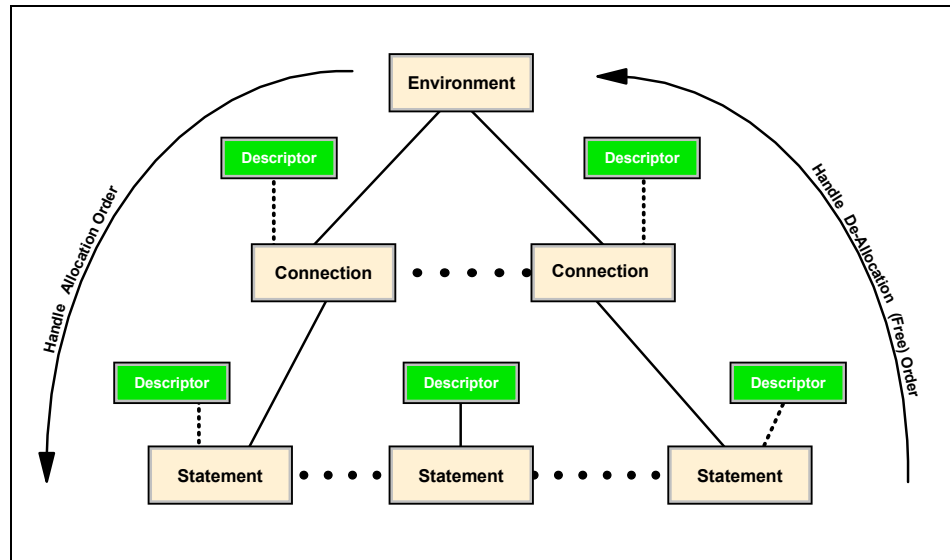


Figure 4-1 CLI handles

4.4.2 The CLI driver

In order for us to use Call Level Interface (CLI) APIs, we need to make use of the CLI driver available in DB2. The name of the library specific to each platform is:

- ▶ On Windows: db2cli.dll (for dynamic loading) or db2cli.lib (for static linking)
- ▶ Linux: libdb2.a (for either dynamic loading at runtime or static linking at compile time)

In this book, we discuss the basic programming features available in the CLI interface.

4.4.3 The CLI configuration file (db2cli.ini)

If we ever need to change the default behavior of CLI, we can change the db2cli.ini configuration file by adding or modifying CLI keywords. This file is read by the CLI driver during runtime, and is either located in the directory specified by the DB2 environment registry setting DB2CLIINIPATH (if set by the user) or in the following directories.

Table 4-4 Locations of the db2cli.ini file

Windows	SqlLib in the DB2 installation path
Linux	sqllib/cfg of the instance owner

An example of the contents of a db2cli.ini file is found in Example 4-28.

Example 4-28 Sample db2cli.ini file contents

```
; Comment lines start with a semi-colon.

[tstcli1x]
uid=userid
pwd=password
autocommit=0
TableType="'TABLE','VIEW','SYSTEM TABLE'"

[tstcli2x]
; Assuming dbalias2 is a database in DB2 for MVS.
SchemaList="'OWNER1','OWNER2',CURRENT SQLID"

[MyVeryLongDBALIASName]
dbalias=dbalias3
SysSchema=MYSHEMA

[testDB]
autocommit=0
```

In a db2cli.ini file, the name of a database is surrounded by square brackets. Each database in the file will have its own section of CLI keywords, which are listed below the database name. If a database is not listed in the db2cli.ini file, this means that there are no CLI keywords associated with it. Comments are denoted by semicolons (;). In Example 4-28, we see that there is a database named testDB with one CLI keyword named autocommit.

If we want a CLI keyword to apply to all databases, instead of defining the keyword in each database section, we can use a common section. A common section (denoted by “[common]”) in the configuration file means that the CLI driver will read the CLI keyword during runtime for all databases it accesses.

We can choose to modify the db2cli.ini manually or on the DB2 Command Line Processor (CLP). To list the entire contents of the db2cli.ini file on the CLP, we can run the command:

```
db2 get cli cfg
```

To only get CLI keywords applicable for a particular database, we can run the following command (where *<database name>* is the name of the database):

```
db2 get cli cfg for section <database name>
```

To add or update a CLI keyword for a particular database, we can run the following command (substituting for the actual database name, CLI keyword, and value):

```
db2 update cli cfg for section <database name> using <CLI keyword>
<value>
```

If a section for a database does not already exist and we use the UPDATE CLI CFG command, a section will be automatically added to the configuration file. An example of adding a CLI keyword to the `db2cli.ini` file on the CLP for the SAMPLE database is:

```
db2 update cli cfg for section sample using autocommit 0
```

Refer to the DB2 UDB *Call Level Interface Guide & Reference*, SC10-4224, for a list of all supported CLI keywords.

4.4.4 Setting up the CLI Environment

In addition to setting up the C/C++ development environment as outlined in 4.1.1, “C/C++ development environment setup” on page 148, programs written using CLI need to ensure that the necessary CLI packages exist on the database server.

The CLI driver will communicate with the application and the database to process SQL statements. Precompile and bind steps are not required for CLI applications. This means that no application packages are created in the database. However, for an SQL statement to be executed, there still needs to be a package with available sections in the database for access plans. This is where CLI bind files come in. CLI bind files are supplied by DB2 and only have to be bound against the database once. Once bound, CLI dynamic placeholder packages will exist in the database. These packages are ready to handle the dynamic SQL passed by the driver.

DB2 supplies text files listing the required bind files to be bound against a particular database server platform. Table 4-5 on page 181 outlines the list files specific to each database server platform. They are available in the BND directory of the DB2 installation path.

Table 4-5 DB2 list files for CLI

Server platform	Needed .lst files for CLI
DB2 UDB (Windows, UNIX)	db2cli.lst, db2ubind.lst
DB2 for z/OS® and OS/390®	ddcsmvs.lst
DB2 for VM	ddcsvm.lst
DB2 for VSE	ddcsvse.lst
DB2 for AS/400® and iSeries™	ddcs400.lst

To bind the packages, you need to connect to the database server and run the BIND command to process the appropriate .lst file.

For example, within a DB2 UDB CLP window on a Windows machine, suppose we want to bind the CLI packages against a DB2 database on a Linux machine. Example 4-29 shows the commands we execute to create the CLI and DB2 utility packages in the SAMPLE database.

Example 4-29 Binding CLI bind files

```
db2 connect to sample
db2 bind C:\Program Files\IBM\SQLLIB\bnd\@db2ubind.lst sqlerror
continue grant public
db2 bind C:\Program Files\IBM\SQLLIB\bnd @db2cli.lst sqlerror continue
grant public
db2 terminate
```

In the example above, you might notice that an at character, (@), is added in front of the list filename. This is needed for the DB2 binder to properly process each bind file listed in the list file.

4.4.5 Overview of steps

The steps to build a C/C++ application using CLI are as follows:

1. Allocate an environment handle.
2. Allocate one or more connection handles to be associated with the single environment handle allocated in step 1.
3. With a connection handle allocated in Step 2, perform a database connection. Allocate one or more statement handles under the connection handle to do SQL processing against the database.
4. Repeat step 3 if there are multiple connection handles.
5. Clean up resources by ensuring all database connections are disconnected and all handles are freed before the application terminates.

4.5 A simple C inventory program using CLI

To illustrate the basic fundamentals of building a C/C++ application using the CLI interface, let us develop a simple program and work with the INVENTORY table in the SAMPLE database again. This application will print a list of products, which are low in quantity, see Example 4-30. The user must supply the minimum acceptable quantity value. Any products with quantity less than the acceptable value need to be restocked.

Example 4-30 Sample output of CLI program

```
CLIinventory 50

Products to Restock (less than 50)
-----
5      100-100-01
25     100-101-01
```

In the above run, the user tells the application that we need a list of all products, which have a quantity of less than 50. The application returns a list of two items in the database satisfying this requirement.

4.5.1 CLI header files

To begin a CLI program, we need to ensure one of the following header files is included:

- ▶ `sqlcli.h` - Contains CLI constants, function prototypes and data structures
- ▶ `sqlcli1.h` - Contains everything in `sqlcli.h` as well as SQL extensions in `sqlxext.h`

4.5.2 Allocating handles

There are various CLI APIs we can use to allocate different types of handles. Refer to DB2 documentation for the syntax of each API. In Table 4-6, we list the APIs.

Table 4-6 APIs to allocate handles

CLI API	Purpose
<code>SQLAllocEnv</code>	Get an environment handle.
<code>SQLAllocConnect</code>	Get a connection handle.
<code>SQLAllocHandle</code>	Get a handle.
<code>SQLAllocStmt</code>	Get a statement handle.

Note: `SQLAllocEnv()`, `SQLAllocConnect()`, and `SQLAllocStmt()` are all ODBC 2.0 APIs and are deprecated. These APIs should be replaced by `SQLAllocHandle`.

We use `SQLAllocHandle` in our sample CLI application.

This is the syntax of the function, as outlined in the DB2 documentation:

```
SQLRETURN SQLAllocHandle (
    SQLSMALLINT      HandleType,      /* fHandleType */
    SQLHANDLE         InputHandle,     /* hInput */
    SQLHANDLE         *OutputHandlePtr; /* *phOutput */
```

In the above syntax, `HandleType` is the type of handle, `InputHandle` is the context for the new handle, and `OutputHandlePtr` is a buffer storing the newly allocated handle data structure.

To begin the application, we need to define and allocate an environment and connection handle. Example 4-31 on page 184 illustrates how to do this. We declare two variables to store the data structure of the handles and then call the `SQLAllocHandle()` API. When environment handles are allocated, there is no

context for that handle type, so we use `SQL_NULL_HANDLE` as the context. When connection handles are allocated, they need to be under the context of an environment handle.

Example 4-31 Using `SQLAllocHandle()`

```
SQLHANDLE env_handle;
SQLHANDLE conn_handle;
...
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env_handle);
...
SQLAllocHandle(SQL_HANDLE_DBC, env_handle, &conn_handle);
```

4.5.3 Freeing handles

At the end of an application, we need to ensure we free the resources associated with each handle. There are various CLI APIs we can use to free up the different types of handles. In Table 4-7, we list the APIs.

Table 4-7 APIs to free handles

CLI API	Purpose
<code>SQLFreeEnv</code>	Free an environment handle.
<code>SQLFreeConnect</code>	Free a connection handle.
<code>SQLFreeStmt</code>	Free a statement handle.
<code>SQLFreeHandle</code>	Free a handle.

Note: `SQLFreeEnv()`, `SQLFreeConnect()`, and `SQLFreeStmt(SQL_DROP)` are deprecated. `SQLFreeHandle()` should be used to free environment, connection, and statement handles.

Generally, we can use the `SQLFreeHandle()` API to free environment, connection, and statement handles. We use `SQLFreeHandle()` API in our sample CLI application. To end our sample application, we need to free the connection and environment handles. We show how to do this with the `SQLFreeHandle()` API, giving a handle type and handle for each call in Example 4-32.

Example 4-32 Using `SQLFreeHandle()`

```
SQLFreeHandle(SQL_HANDLE_DBC, conn_handle);
SQLFreeHandle(SQL_HANDLE_ENV, env_handle);
```

4.5.4 Connecting and disconnect to and from a database

Table 4-8 lists some APIs that can be used in working with a connection to a database.

Table 4-8 Connection-related CLI APIs

CLI API	Purpose
SQLConnect	Connect to a database given a database name, user ID, and password.
SQLDriverConnect	Connect to a database (has expanded connection parameters).
SQLBrowseConnect	Use iterative method to connect to a database.
SQLSetConnectAttr	Set connection attributes.
SQLSetConnection	Sets the current active connection. Used for embedded SQL modules within a CLI application.
SQLGetConnectAttr	Get connection option value.
SQLDisconnect	Disconnect from the database.

There are various APIs to choose from for connecting to a database. The `SQLDriverConnect()` API expands the functionality of `SQLConnect()` by allowing extra connection parameters and the ability to get connection information from the user. The `SQLBrowseConnect()` is an iterative way of connecting to the database. In our application, we use the `SQLConnect()` API:

```
SQLRETURN SQLConnect (
    SQLHDBC          ConnectionHandle,      /* hdbc */
    SQLCHAR          *ServerName,          /* szDSN */
    SQLSMALLINT      ServerNameLength,     /* cbDSN */
    SQLCHAR          *UserName,           /* szUID */
    SQLSMALLINT      UserNameLength,      /* cbUID */
    SQLCHAR          *Authentication,     /* szAuthStr */
    SQLSMALLINT      AuthenticationLength); /* cbAuthStr */
```

In our code, we add the following line (Example 4-33 on page 186) to establish a default database connection and later to disconnect from the database.

Example 4-33 Using SQLConnect() & SQLDisconnect()

```
SQLConnect(conn_handle, "sample", SQL_NTS, NULL, SQL_NTS, NULL,  
SQL_NTS);  
...  
SQLDisconnect(conn_handle);
```

4.5.5 Processing SQL statements

After connecting to the database, we need to allocate a statement handle to manage our SQL statement. Once this is done, we can execute an SQL statement by either preparing the statement or executing the statement directly. Figure 4-2 illustrates this process.

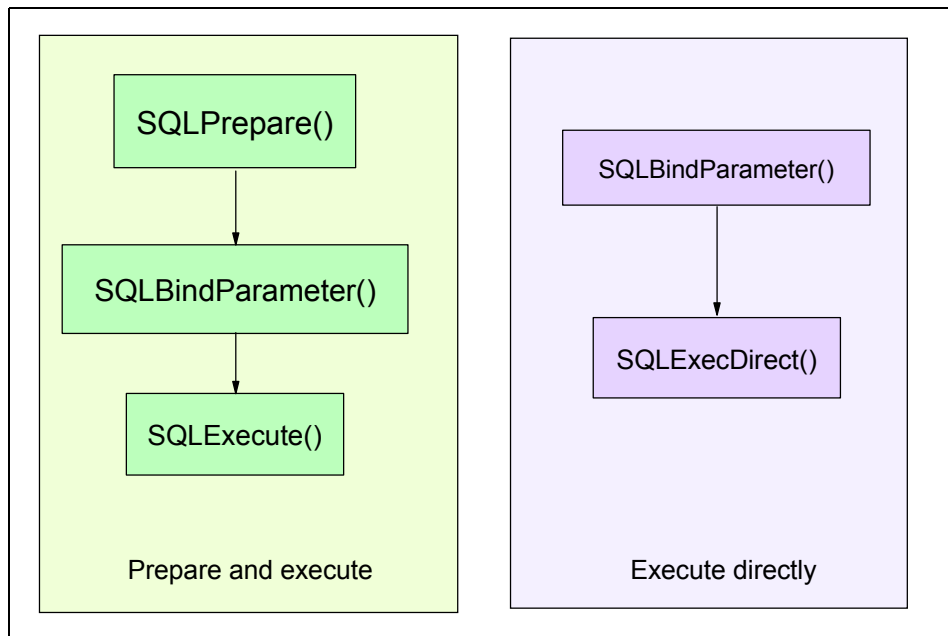


Figure 4-2 Processing SQL statements

In either case, if parameter markers exist in the SQL statement, `SQLBindParameter()` must be called to associate each parameter marker to an application buffer or a LOB locator. If a `SELECT` statement is run and a result set expected, a cursor is automatically opened. So, unlike embedded SQL, cursors neither have to be declared nor opened. To fetch the results, `SQLBindCol()` is first called to bind the columns of a result set to application variables, then `SQLFetch()` is called to fetch the rows.

After the processing of an SQL statement is complete, the statement handle can be freed. In the complete CLI code definition in Example 4-34 on page 188, we show how to prepare, bind, and execute a SELECT statement and fetch the results. Table 4-9 lists some CLI APIs that you can use for processing an SQL statement.

Table 4-9 Some CLI APIs for SQL processing

CLI API	Purpose
SQLPrepare	Prepare an SQL statement.
SQLExtendedPrepare	Prepare an array of statement attributes for an SQL statement.
SQLExtendedBind	Bind an array of columns.
SQLBindParameter SQLSetParam	Bind a parameter marker in an SQL statement.
SQLDescribeParam	Get info about a parameter marker.
SQLExecute	Execute a prepared statement.
SQLExecDirect	Execute a statement.
SQLNumParams	Get the number of parameters in a statement.
SQLRowCount	Get the number of rows affected by an insert/update/delete or number of rows in a result set.
SQLNumResultCols	Get the number of columns in a result set.
SQLDescribeCol	Describe a column in the result set.
SQLColAttribute SQLColAttributes	Get information about the attributes of a column in a result set.
SQLBindCol	Bind a column in the result set to an application variable.
SQLFetch	Get a result set row.
SQLExtendedFetch	Get multiple result set rows.
SQLCancel	Cancel an SQL statement.
SQLTransact SQLCloseCursor	Commit or roll back a transaction.
SQLEndTran	End a transaction.

4.5.6 Complete CLI Inventory Program

In our example CLI inventory program (Example 4-34), we connect to the database, prepare a SELECT statement, and then bind the parameter marker. Because QUANTITY is of type INTEGER in the database, we bind the parameter marker with a C type of SQL_C_LONG and SQL type of SQL_INTEGER. In binding a parameter marker, the SQL data type and the symbolic C data type need to be known as input to the CLI API.

We also call `SQLBindCol()` for each column of the result set. Because QUANTITY and PID data values are returned from the INVENTORY table, we call `SQLBindCol()` to bind both columns to application buffers. Once we finish processing the result set, we disconnect from the database.

For simplicity and ease of readability, we have omitted error checking in our example CLI application.

Example 4-34 CLInventory.c

```
/* Source File Name: CLInventory.c
**
** This simple CLI program will print a list of
** products that need to be restocked.
** The user needs to supply the minimum acceptable
** quantity value before a product is considered to be
** low in stock.
**
***/
#include <stdlib.h>
#include <stdio.h>
#include <sqlcli.h>

int main(int argc, char *argv[]) {
    SQLHANDLE env_handle;
    SQLHANDLE conn_handle;
    SQLHANDLE stmt_handle;
    SQLCHAR PID[11];
    SQLINTEGER quantity;
    SQLLEN ind[2];
    SQLINTEGER min_quality=atoi(argv[1]);

    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env_handle);
    SQLAllocHandle(SQL_HANDLE_DBC, env_handle, &conn_handle);
```

```

    SQLConnect(conn_handle, "sample", SQL_NTS, NULL, SQL_NTS, NULL,
SQL_NTS);
    SQLAllocHandle(SQL_HANDLE_STMT, conn_handle, &stmt_handle);
    SQLPrepare(stmt_handle, "select quantity,PID from inventory where
quantity <? order by quantity", SQL_NTS);
    SQLBindParameter(stmt_handle, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER,0,0,&min_quality,0,NULL);
    SQLExecute(stmt_handle);
    SQLBindCol(stmt_handle, 1, SQL_C_LONG, &quantity, 0,&ind[0]);
    SQLBindCol(stmt_handle, 2, SQL_C_CHAR, PID, sizeof(PID),&ind[1]);

    printf("\nProducts to Restock (less than %d)\n",min_quality);
    printf("-----\n");
    while (SQLFetch(stmt_handle)==SQL_SUCCESS)
        printf("%d\t%s\n",quantity, PID);

    SQLFreeHandle(SQL_HANDLE_STMT, stmt_handle);
    SQLDisconnect(conn_handle);
    SQLFreeHandle(SQL_HANDLE_DBC,conn_handle);
    SQLFreeHandle(SQL_HANDLE_ENV,env_handle);
    return SQL_SUCCESS;
}

```

4.5.7 Error handling

After each API call, we should be checking the return code status to ensure that no errors have occurred. Table 4-10 on page 190 lists possible CLI function return codes.

Table 4-10 CLI function return codes

CLI function return code	CLI function result
SQL_SUCCESS	Successful. No SQLSTATE information available.
SQL_SUCCESS_WITH_INFO	Successful, but some informational message or warning was returned.
SQL_STILL_EXECUTING	Still executing although control has been returned to the application.
SQL_NO_DATA_FOUND	Successful, but no data was found.
SQL_NEED_DATA	Missing parameter data needed for the execution of an SQL statement.
SQL_ERROR	Error.
SQL_INVALID_HANDLE	Error. Invalid handle specified by the application

If a function call resulted in an unexpected return code, diagnostic records are produced and associated with the handle that executed the API. To retrieve the diagnostic information, applications can make use of functions as shown in Table 4-11.

Table 4-11 Some diagnostic CLI APIs

CLI API	Purpose
SQLError	Get information on an error.
SQLGetDiagField	Get a field in a diagnostic record.
SQLGetDiagRec	Get diagnostic record.

4.5.8 Quick SQLGetDiagRec() example

Let us see an example of how to use the SQLGetDiagRec() API. We first define the variables within a CLI application needed for the function call:

```
SQLCHAR SQLState[6]; // Will store the SQLState
SQLCHAR msgText[SQL_MAX_MESSAGE_LENGTH]; // Will store the error msg
text
SQLINTEGER nativeErrorCode; // Will store the native error code
SQLSMALLINT msgTextLength; // Will store the error msg text length
```

Now let us try connecting to an invalid database and see what is provided by the diagnostic function. We run a program with the following lines:

```
SQLConnect(conn_handle, "samdple", SQL_NTS, NULL, SQL_NTS, NULL,  
SQL_NTS);
```

```
SQLGetDiagRec(SQL_HANDLE_DBC, conn_handle, 1, SQLState,  
&nativeErrorCode, msgText, sizeof(msgText), &msgTextLength);
```

```
printf("SQLState: %s\nNative Error: %d\nError Message Text:  
%s", SQLState, nativeErrorCode, msgText);
```

This produces the following output to let us know the database name was incorrectly specified:

```
SQLState: 08001  
Native Error: -1013  
Error Message Text: [IBM][CLI Driver] SQL1013N The database alias name  
or database name "SAMDPLE" could not be found.  SQLSTATE=42705
```

4.6 XML support

The XML data type is supported for use in embedded SQL and CLI applications. We briefly discuss how to work with the data type within C/C++ applications.

4.6.1 Embedded SQL

Embedded applications can make use of XML, LOB, or LOB_FILE data types when working with XML data in a DB2 database. If you choose to use XML host variables, they will be implicitly parsed, whereas using host variables of character and binary data type might not be. For dynamic SQL, CLOB, and BLOB are also implicitly parsed. For static SQL, an explicit XMLPARSE() will be injected in the SQL statement, but only for CLOB and BLOB (not DBCLOB). To declare host variables to handle XML data, we do so in the DECLARE section as we normally would for host variables of other data types. For XML host variables, we can use declarations of the form:

```
SQL TYPE IS XML AS <base type> <host var>
```

In the above statement, <host var> denotes the host variable name, and <base type> denotes the data type of the XML host variable. The possible values are listed in Table 4-12 on page 192.

Table 4-12 Allowed XML host variable data types

Data type	XML data encoding
CLOB(<i>n</i>)	Application mixed code page stored in CLOB host variable.
DBCLOB(<i>n</i>)	Application graphic code page stored in DBCLOB host variable.
BLOB(<i>n</i>)	Internally encoded in a BLOB host variable.
CLOB_FILE	Application mixed code page stored in CLOB file.
DBCLOB_FILE	Application graphic code page stored in DBCLOB file.
BLOB_FILE	Internally encoded in a BLOB file.

Once the host variable declarations are done, we can write code to process SQL statements using XML data in the same way we write code to process other types of SQL statements. Refer to Chapter 2, “Application development with DB2 pureXML” on page 49 to obtain information about how to write SQL statements to interact with data of the DB2 XML data type.

Selecting XML data using embedded SQL

Example 4-35 below shows a simple example of how to retrieve an XML column in embedded SQL.

Example 4-35 Retrieving an XML column using embedded SQL

```
EXEC SQL BEGIN DECLARE SECTION;
    SQL TYPE IS CLOB(10K) clob1;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL CONNECT TO sample;
...
EXEC SQL SELECT description INTO :xmlClob1 FROM product WHERE
PID='100-100-01';
printf("XML data length: %d\nXML data: %s",clob1.length,clob1.data);
EXEC SQL COMMIT;
EXEC SQL CONNECT RESET;
```

In Example 4-35, we first declare a host variable of character LOB type to hold the serialized string format of our XML data. The precompile form of the clob1 host variable shows that clob1 is a structure with a length and data field:

```
struct clob1_t {
    sqluint32 length;;
    char data[10240];
} clob1;
```


We then run a select statement to retrieve the DESCRIPTION field of the PRODUCT table in the SAMPLE database. In the database, the DESCRIPTION column is of type xml. We call the XMLSERIALIZE method to convert the XML data from the database hierarchical format to an application CLOB and store the result in clob1. Running the application produces the following output.

Example 4-36 Output from running Example 4-35 on page 192

```
XML data length: 251
XML data: <product xmlns="http://posample.org"
pid="100-100-01"><description><name>Snow Shovel, Basic 22
inch</name><details>Basic Snow Shovel, 22 inches wide,
straight handle with D-Grip</details><price>9.99</price><weight>1
kg</weight></description></product>
```

Inserting XML data using embedded SQL

Example 4-37 shows a simple example of how to insert XML data into the XML column (DESCRIPTION column in the PRODUCT table) using embedded SQL.

Example 4-37 Insert data into XML column

```
#include <sqlenv.h>
#include <sqlutil.h>

EXEC SQL INCLUDE SQLCA;

EXEC SQL BEGIN DECLARE SECTION;
    char xmldata[1000];
    short nullind;
    static SQL TYPE IS XML AS CLOB(1k) xmlclob1=SQL_CLOB_INIT(" ") ;

EXEC SQL END DECLARE SECTION;

int main(int argc, char *argv[])
{
    nullind = 0;
    /* Create a XML document for Description column*/
    strcpy (xmldata, "<product xmlns=\"http://posample.org\"
pid=\"102-102-02\">"
           "<description><name> Shovel </name>"
           "<details>Basic Shovel, 22 inches wide, straight
handle with D-Grip</details>"
           "<price>19.99</price>"
           "<weight>1.5 kg</weight>"
           "</description></product>");
```

```

/* description and product */
strcpy(xmlclob1.data, xmldata);

/* Set the length of the data */
xmlclob1.length = strlen(xmldata) + 1;

/* connect to database */
EXEC SQL CONNECT TO sample;

/* inserting when source is from host variable of type XML AS CLOB */
printf(" Inserting when source is from host variable of type XML AS
CLOB\n");
EXEC SQL INSERT INTO product (pid, DESCRIPTION)
VALUES ('102-102-02', :xmlclob1:nullind);

if (sqlca.sqlcode != 0)
{
    printf("\n Insertion failed\n");
    printf(" FAILED WITH SQLCODE = %d\n\n", sqlca.sqlcode);
}
EXEC SQL COMMIT;
/* disconnect from the database */
EXEC SQL CONNECT RESET;

return 0;
} /* main */

```

In Example 4-37 on page 193, we declare a host variable of XML as CLOB. The bind file resulting from the db2 prep utility results in the following **db2bfd -v** command output:

```

InsertProd.bnd: Host Variables = 3
Type SQL Data Type Length Alias Name_Len Name UDT Name
-----
460 C STRING 1000 H00001 7 xmldata
500 SMALLINT 2 H00002 7 nullind
408 CLOB 1024 H00003 8 xmlclob1 XML

```

Updating XML data column using embedded SQL

Example 4-38 on page 195 below shows a simple example of how to update XML data in the XML column (*DESCRIPTION* column of the *PRODUCT* table) using embedded SQL.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlenv.h>
#include <sqlutil.h>

EXEC SQL INCLUDE SQLCA;

EXEC SQL BEGIN DECLARE SECTION;
    char xmldata[2000];
    char parse_option[30];
    static SQL TYPE IS XML AS CLOB(1k) xmlclob1=SQL_CLOB_INIT(" ") ;
EXEC SQL END DECLARE SECTION;

int main(int argc, char *argv[])
{
    int rc = 0;

    /* Create a XML document that will be used to INSERT in the table */
    strcpy(xmldata, "<product xmlns=\"http://posample.org\"
pid=\"102-102-02\">"
        "<description><name> Shovel </name>"
        "<details>Basic Shovel, 22 inches wide, straight
handle with D-Grip</details>"
        "<price>9.99</price>"
        "<weight>1.5 kg</weight>"
        "</description></product>");

    strcpy(xmlclob1.data, xmldata);

    xmlclob1.length = strlen(xmldata) + 1;

    /* check the command line arguments */
    EXEC SQL CONNECT TO sample;

    /* Update the XML column using host variable of type XML */
    printf(" Update XML column using variable of type XML\n");
    EXEC SQL UPDATE product SET DESCRIPTION = :xmlclob1 WHERE pid =
'102-102-02';

    if (sqlca.sqlcode != 0)
    {
        printf("\nUpdate failed\n");
    }
}
```

```

        printf(" FAILED WITH SQLCODE = %d\n\n", sqlca.sqlcode);
    }
EXEC SQL COMMIT;
/* disconnect from the database */
EXEC SQL CONNECT RESET;

} /* main */

```

In Example 4-38 on page 195, we declare a host variable of XML as CLOB. The bind file resulting from the db2 prep utility results in the following **db2bfd -v** command output:

updtProd.bnd: Host Variables = 3

Type	SQL Data Type	Length	Alias	Name_Len	Name	UDT Name
460	C STRING	2000	H00001	7	xmldata	
460	C STRING	30	H00002	12	parse_option	
408	CLOB	1024	H00003	8	xmlclob1	XML

4.6.2 Call Level Interface (CLI)

CLI Applications can use the xml, binary, or character data types to work with XML data from a DB2 database. In CLI API calls, applications can use application C types of the following:

- ▶ SQL_C_BINARY
- ▶ SQL_C_CHAR
- ▶ SQL_C_WCHAR
- ▶ SQL_C_DBCHAR

This means that when applications work with XML data types, they can bind application variables to any of the types just mentioned. The default C type is SQL_C_BINARY and using this type can avoid code page conversion issues. The other C types assume the application code page encoding. That is, SQL_C_BINARY would output in UTF-8, SQL_C_WCHAR in platform-encoding UTF-16, and SQL_C_CHAR/SQL_C_DBCHAR in the application mixed and graphic code pages respectively.

On the database end, a column of type xml has a symbolic SQL type of SQL_XML. The application can use this SQL type when trying to store or retrieve XML data.

Let us try a simple CLI example. Assuming we have initialized all the necessary data structures and are connected to the database, we now want to get back the

DESCRIPTION column from the PRODUCT table. The column in the database is of type xml. We can do the following (Example 4-39).

Example 4-39 Retrieving an XML column using CLI

```
SQLCHAR description[10001];
SQLINTEGER ind;
...
SQLAllocHandle(SQL_HANDLE_STMT, conn_handle, &stmt_handle);
...
SQLExecDirect(stmt_handle, "select description from product where
PID='100-100-01'", SQL_NTS);
SQLBindCol(stmt_handle, 1, SQL_C_CHAR, description,
sizeof(description), &ind);
SQLFetch(stmt_handle);
printf("%s",description);
...

```

In Example 4-39, we declare a buffer called DESCRIPTION to hold the returned value from the database. We call `SQLExecDirect()` to execute the `SELECT` statement, and call `SQLBindCol()` to bind the returned column value to the application storage buffer named `description`. We specify `SQL_C_CHAR` as the C data type because we are expecting the xml to be converted to a CLOB in our `XMLSERIALIZE` call. Running the program produces the output as shown in Example 4-40.

Example 4-40 Output from running Example 4-39

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<product xmlns="http://posample.org"
pid="100-100-01"><description><name>Snow Shovel, Basic 22
inch</name><details>Basic Snow Shovel, 22 inches wide, straight handle
with D-Grip</details><price>9.99</price><weight>1
kg</weight></description></product>

```



Application development with Java

In this chapter, we describe how to develop a Java application with DB2 Express-C. This chapter contains the following:

- ▶ Application software requirements
- ▶ In-depth description of the `java.sql` package support in DB2 Type 4 driver, including stored procedure and transactions
- ▶ An overview of `javax.sql` package
- ▶ Exception handling
- ▶ XML/XQuery support
- ▶ SQLj support

5.1 Application requirements

Any Java application needs the following packages to access the DB2 JDBC universal driver and use it to make the connection to the DB2 database:

- ▶ `java.sql` contains the core JDBC API.
- ▶ `com.ibm.db2.jcc` contains the DB2-specific implementation of JDBC for universal JDBC driver. `db2jcc.jar` archive contains this package.

For accessing the `DataSource` object for enterprise application, you need the following packages along with the above:

- ▶ `javax.naming` contains classes and interfaces for Java Naming and Directory Interface.
- ▶ `javax.sql` contained JDBC 2.0 standard extensions (`db2jcc_javax.jar`).
- ▶ `javax.transactions` contains support for distributed transaction for DB2 JDBC Type 2 driver.

Along with this files, one of these two files should be included in CLASSPATH:

- ▶ `db2jcc_license_cu.jar`
- ▶ `db2jcc_license_cisuz.jar`

5.2 Drivers

Java applications, which use DB2 as a database, connect to the database using different kinds of drivers. According to the JDBC specification, there are four types of driver architecture. DB2 supports Type 2 (deprecated as of DB2 V8.2) and a combination of Type 2 and Type 4 drivers called *universal JDBC drivers*, which combine the Type 2 and Type 4 functionalities. Using universal driver, an application can utilize both Type 2 and Type 4 functionalities using a single instance of driver in memory.

Universal JDBC driver for DB2

Whenever an application loads the universal driver, a single instance of Type 2 and Type 4 driver is loaded in memory and can be used to make Type 2 and Type 4 connections. This driver is an entirely new driver and the behavior of this driver might be different from the standard Type 2 and Type 4 drivers. The Type 2 and Type 4 driver connectivity depend on the type of URL provided to the driver while getting the connection.

For Type 2 driver connectivity, use the following URL format:

```
jdbc:db2:database name
```


For Type 4 driver connectivity, use the following URL format:

```
jdbc:db2://host name:port number/database name
```

The db2jcc.jar package contains the universal driver classes.

5.3 Application example

In this chapter, we discuss the fundamentals of application development using DB2 JDBC and guide you through the Java application development process using a small shopping cart application. This application uses the SAMPLE database shipped along with the DB2 Express-C. The application does the following:

- ▶ Takes the customer ID as an input. Checks if the customer exists. If the customer exists, the application goes to the next page to take the order.
- ▶ Displays all the available products. The user can select a product from the list and add it to the cart.
- ▶ While checking out, the application gives the purchase order ID and inserts the purchase order details in the table.

The class in Example 5-1 on page 202 has some of the support methods used for the application. We use this example along with others to explain how to access DB2 data. At the end of this chapter, you can follow the steps provided to run the application. You can find the sample application download procedure at Appendix C, “Additional material” on page 319.

This class contains the following method:

- ▶ `isCustomer(int)`:
This method checks to see if the customer ID passed to the method as an argument exists in the database or not. It returns the customer name if the id exists.
- ▶ `getProducts()`:
This method returns the name of all the available products.
- ▶ `getDescription(String)`:
This method returns the description of the product id passed as an argument.
- ▶ `createCart(HashMap)`:
This creates a cart table to store the temporary data for the cart.
- ▶ `updateTable()`:
Update the tables with the purchase order details.

- ▶ `createPorder(poid)`:
This calls a stored procedure to create the XML value.

Example 5-1 OrderProcess class

```
import java.sql.*;
import java.util.*;
import com.ibm.db2.jcc.*;
public class OrderProcess {

    private Connection con=null;
    private String[] products=null;
    private int cid=0;
    public static void main(String args[])
    {
        String[] products=null;
        OrderProcess op=new OrderProcess();
        int custid=1001;
        HashMap hm=new HashMap();
        hm.put("Snow Shovel, Deluxe 24 inch",4);
        hm.put("Snow Shovel, Basic 22 inch",5);
        op.getConnection();
        products=op.getProducts();
        for(int i=0;i<products.length && products[i]!=null ;i++)
        {
            System.out.println(products[i]);
        }

        String name=op.isCustomer(custid);
        if(name!=null)
        {
            op.dropCart();
            op.createCart(hm);
            Double price = op.findTotal();
            System.out.println("your total is " +price);
            op.updateTable();
        }

    }

    public void getConnection()
    {
        try {
            if(con==null)
            {
```

```

        Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();

con=DriverManager.getConnection("jdbc:db2://localhost:50000/sample","user1","db2usrpw");
        products=getProducts();

        }
        System.out.println(con);
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        System.out.println(e.getErrorCode());
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

// isCustomer method will check if the customer with the id exists
// in the customer table or not. The function returns the name of
// the customer if the customer exists; otherwise, it returns null
public String isCustomer(int id)
{
    String query="select xmlquery('declare default element namespace
\"http://posample.org\"; $id/customerinfo/name' passing by ref
customer.info as \"id\") from customer where cid=?";
    try {
        PreparedStatement stmt=con.prepareStatement(query);
        stmt.setInt(1,id);
        ResultSet rs=stmt.executeQuery();
        cid=id;
        if(!rs.next())
            return null;
        else
            return rs.getString(1);
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        System.out.println(e.getErrorCode());
        e.printStackTrace();
        return null;
    }
}

} // isCustomer

// getProduct returns all the product names
public String[] getProducts()

```

```

    {
    String[] products=new String[10];
    int i=0;
    String query="select name from product";
    Statement stmt;
    try {
        stmt = con.createStatement();
    ResultSet rs=stmt.executeQuery(query);
    while(rs.next())
        {
            products[i]=rs.getString(1);
            i++;
        }
    return products;
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        System.out.println(e.getErrorCode());
        e.printStackTrace();
        return null;
    }
} // getProducts

// getDescription method returns the description of the product
public Object getDescription(String name)
{
    try {
        PreparedStatement stmt=con.prepareStatement("select description
from product where name=?");
        stmt.setString(1,name);
        ResultSet rs=stmt.executeQuery();
        rs.next();
        String
value=((com.ibm.db2.jcc.DB2Xml)rs.getObject(1)).getDB2String();
        return value;
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        System.out.println(e.getErrorCode());
        e.printStackTrace();
        return null;
    }
} // getDescription

public void dropCart()
{
    String cart = "cart"+cid;

```

```

try {
Statement stmt=con.createStatement();
stmt.executeUpdate("drop table " + cart);
}catch (SQLException e) {
    System.out.println(e.getMessage());
    System.out.println(e.getErrorCode());
    e.printStackTrace();
}
}
//createCart method creates a table to store the product id
//and the quantity ordered by the customer temporarily. This
//table is used to create the purchaseorder in XML format

public void createCart(HashMap hm)
{
    try {
        Statement stmt=con.createStatement();

        String cart = "cart"+cid;
        String insertQuery="insert into "+cart+ " values((select pid
from product where name=?),?)";
        String createQuery="create table "+cart+"(pid varchar(10),
quantity int)";
        PreparedStatement pstmt=con.prepareStatement(insertQuery);
        System.out.println(insertQuery);
        stmt.executeUpdate(createQuery);
        con.setAutoCommit(false);
        Set set=hm.keySet();
        Iterator it=set.iterator();
        while(it.hasNext())
        {
            String key=(String) it.next();
            pstmt.setString(1,key);
            pstmt.setInt(2,((Integer)hm.get(key)).intValue());
            pstmt.addBatch();
            System.out.println(key +"
"+((Integer)hm.get(key)).intValue());
        }
        pstmt.executeBatch();
        con.commit();
        con.setAutoCommit(true);
    }catch (BatchUpdateException buex) {
        int [] updateCounts = buex.getUpdateCounts();
        for (int i = 0; i < updateCounts.length; i++) {

```

```

        System.err.println(" Statement " + i + ":" +
updateCounts[i]);
        System.err.println(" Message: " + buex.getMessage());
        System.err.println(" SQLSTATE: " + buex.getSQLState());
        System.err.println(" Error code: " + buex.getErrorCode());
        SQLException ex = buex.getNextException();
        while (ex != null) {
            System.err.println("SQL exception:");
            System.err.println(" Message: " + ex.getMessage());
            System.err.println(" SQLSTATE: " + ex.getSQLState());
            System.err.println(" Error code: " + ex.getErrorCode());
            ex = ex.getNextException();
        }
    } // createCart

public double findTotal()
{
    Statement stmt;
    try {
        stmt = con.createStatement();
        String query="select sum(price*quantity) from cart"+cid+",
product where cart"+cid+".pid=product.pid";
        ResultSet rs= stmt.executeQuery(query);
        rs.next();
        return rs.getDouble(1);
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        System.out.println(e.getErrorCode());
        e.printStackTrace();
        return 0;
    }
} // updateTable updates the purchaseorder table with the new purchase
// order
public int updateTable()
{
    try {
        Statement stmt=con.createStatement();
        String porder=null;
        String dropQuery="drop table cart"+cid;
        con.setAutoCommit(false);
        ResultSet rs=stmt.executeQuery("select max(poid) from
purchaseorder");
        rs.next();

```

```

        int poid=rs.getInt(1)+1;
        porder=createPorder(poid);

        System.out.println(porder);
        PreparedStatement pstmt=con.prepareStatement("Insert into
purchaseorder(poid, status, custid, porder)
values(?,?,?,XMLPARSE(document cast(? as varchar(5000))))");
        pstmt.setInt(1,poid);
        pstmt.setString(2,"unshipped");
        pstmt.setInt(3,cid);
        pstmt.setString(4,porder);
        pstmt.executeUpdate();
        con.commit();
        con.close();;
        return poid;
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        System.out.println(e.getErrorCode());
        e.printStackTrace();
        return 0;
    }
} // updateTable

// createPorder calls a stored procedure to create the XML
// document for porder XML column value
public String createPorder(int poid)
{
    CallableStatement cstmt;
    String cart="cart"+cid;
    try {
        cstmt = con.prepareCall("call createOrder(?,?,?)");

        cstmt.setInt(1,poid);
        cstmt.setString(2,card);
        cstmt.registerOutParameter(3, Types.VARCHAR);
        cstmt.executeUpdate();
        return cstmt.getString(3);
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        System.out.println(e.getErrorCode());
        e.printStackTrace();
        return null;
    }
} // createPorder

```

```
} // OrderProcess
```

At the end of the chapter, we use this class to make a Web application which will run on the server and implement a very simple application to create a purchase order for a shopping cart. This application can also be run as a stand-alone application. To run the application stand-alone, refer to 5.13, “Running the application” on page 247.

5.4 java.sql package

Java.sql package defines the classes and interfaces required for the JDBC program to access the relation data stored in a database. These APIs can be used to connect to the relational database and manipulate the data (insert, update, delete, and so on) stored in tabular form using the SQL standard. The interfaces defined in this package are implemented by the driver specific classes and the definition can differ from vendor to vendor.

5.4.1 Getting a connection

A connection to a database can be obtained using the DriverManager class of the java.sql package.

Before getting connection, the driver specific classes must be loaded and registered to the DriverManager. Any number of drivers can be loaded and registered with the DriverManager. You can use the code shown in Example 5-2 on page 208 to load a driver specific class.

Example 5-2 Loading the Type 4 driver classes

```
Class.forName("com.ibm.db2.jcc.DB2Driver")
```

The forName method take a string argument whose value is the name of the package which implements the interfaces defined in java.sql package.

The connection to a database can be obtained by calling the getConnection method of DriverManager class. This method takes a string value (URL) as an input, which gives the information required to connect to the database. A typical URL format for Type 4 driver is:

```
jdbc:db2://<servername>:<port number>/<database name>
```

The code in Example 5-3 returns the connection as Connection class object.

Example 5-3 Loading the Type 4 driver and getting connection to the database

```
Connection con = null;
Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
con =
DriverManager.getConnection("jdbc:db2://localhost:50000/sample","user",
"password");
```

In the case where multiple drivers are loaded, DriverManager is responsible for making use of the appropriate driver to make a connection.

5.4.2 Manipulating data

After getting the connection, data can be selected, inserted, updated, or deleted from the relational tables using SQL statements. JDBC driver implements two interfaces Statement and PreparedStatement for this purpose. An object of any of these classes is required for running an SQL statement.

Statement

An object of Statement (or class implementing the Statement interface) can be used to execute the SQL statement which does not contain parameter markers. An object can be created from the Connection object using createStatement method.

Any number of statements can be created for a particular connection object.

Statement interface defines executeQuery and executeUpdate methods to execute a query statement. The executeQuery method is used when the result set is expected (for example, for the SELECT statement) as output of the query. Alternatively, executeUpdate method is used for updating the database contents (for example, INSERT, UPDATE, and DELETE statements). The executeQuery method returns the ResultSet object, which represents a set of rows returned by the SELECT query. This ResultSet object can be used to fetch the result row by row. executeUpdate returns an integer value, which indicates the number of rows updated, inserted, or deleted from the database based on the type of SQL statement.

SELECT using Statement object

Example 5-4 contains a code snippet for the method getProducts from the application code. The method select the product names from the CUSTOMER table and stores them in the String array.

Example 5-4 SELECT using Statement object

```
String[] products=new String[10];
```

```

int i=0;
String query="get name of product";
Statement stmt;
try {
    stmt = con.createStatement();
    ResultSet rs=stmt.executeQuery(query);
    while(rs.next())
    {
        products[i]=rs.getString(1);
        i++;
    }
}

```

The Statement object also provides execute methods to execute any type of query. The execute method is useful when the decision on SQL statement type is taken at runtime. It returns `true` if the result of the SQL statement is a ResultSet object (for example, for SELECT statement) and `false` if the result is update count or there is no result. Based on the return value, `getResultSet` and `getUpdateCount` methods can be used to get the resultSet object or the update count.

Example 5-5 on page 210 uses the execute method to implement the same function in Example 5-4.

Example 5-5 SELECT using Statement object's execute method

```

String queryStmt="select name from product";
Statement stmt=con.createStatement();
boolean test=stmt.execute(queryStmt);
if (test=true)
    ResultSet rs=stmt.getResultSet()
else
    int count=stmt.getUpdateCount();

```

UPDATE using Statement object

Example 5-6 give the code snippet to update a value in the table using the Statement object.

Example 5-6 UPDATE using Statement object

```

Statement stmt=null;
stmt = con.createStatement();
stmt.executeUpdate("update purchaseorder set status='shipped' where
poid=5007

```

An INSERT or DELETE statement can be executed using the Statement object in the same fashion.

PreparedStatement

An object of PreparedStatement (or a class implementing the PreparedStatement interface) can be used to run the queries, which can contain parameter markers. A PreparedStatement object can be created using the prepareStatement method of Connection object. PreparedStatement extends the Statement interface.

If the SQL statement contains parameter markers, the values for these parameter markers need to be set before executing the statement. Value can be set using setXXX methods of PreparedStatement object where XXX denoted the data type of the parameter marker. setXXX methods are also called *setter* methods.

The following are the examples of setXXX methods:

- ▶ setInt
- ▶ setString
- ▶ setDouble
- ▶ setBytes
- ▶ setClob
- ▶ setBlob

After setting the parameter values, the SQL statement can be executed using any of the executeQuery, executeUpdate, or execute method based on the SQL type.

SELECT using PreparedStatement object

Example 5-7 gives the code snippet for the method isCustomer from the application code. The method passes the value of the customer id to the method, which is used to pass the value to the SELECT query. The method returns null if the customer id does not exist.

Example 5-7 SELECT using PreparedStatement object

```
String query="select info from customer where cid=?";
PreparedStatement stmt=con.prepareStatement(query);
stmt.setInt(1,id);
ResultSet rs=stmt.executeQuery();
if(!rs.next())
return null;
else
return rs.getString(1);
```

INSERT using PreparedStatement

Example 5-8 shows how to insert the value in the table using the PreparedStatement object.

Example 5-8 INSERT using the PreparedStatement object

```
PreparedStatement stmt=null;
stmt = con.prepareStatement("insert into purchaseorder(poid, status)
values(?,?)");
stmt.setInt(1,5020);
stmt.setString(2,"shipped");
stmt.executeUpdate();
|
```

The DELETE and UPDATE statements can be executed in the same fashion.

Tip: Trying to run a query which updates the content of the database with the executeQuery method will result in an exception. Similarly, trying to run a select query using the executeUpdate method will give an exception. Use the execute method whenever you are unsure about the query at compile time (that is, the query is generated at runtime using user's input) and check for the Boolean value returned by the method to decide the result.

CallableStatement

An object of the CallableStatement interface (or the class implementing the CallableStatement interface) can be used to call the stored procedure. The CallableStatement interface extends the PreparedStatement interface. An object of CallableStatement can be created using the prepareCall method of Connection object.

The parameter for a CallableStatement can be of three types:

- ▶ IN
- ▶ OUT
- ▶ INOUT

The value for IN and INOUT parameters must be set before executing the CallableStatement. In the same way, OUT and INOUT parameters should be registered to the database before executing the statement. The CallableStatement can be executed using execute, executeQuery, and executeUpdate methods. The usage of these three methods is:

- ▶ execute: Use this method when multiple result sets are expected as an output.

- ▶ `executeQuery`: Use this method when a single result set is expected.
- ▶ `executeUpdate`: Use this method when no result set is expected.

Example 5-9 gives the code snippet from the `createPorder` method of the application code. The method calls a stored procedure named `createOrder`. The stored procedure code is given in the class `CreateOrder`. The stored procedure has two IN parameters and one OUT parameter. The stored procedure should be registered before using it. 5.5, “Stored procedure support” on page 219 explains how to register the stored procedure.

Example 5-9 Calling a stored procedure using CallableStatement object

```
CallableStatement cstmt;  
cstmt = con.prepareStatement("call createOrder(?,?,?)");  
cstmt.setInt(1,poid);  
cstmt.setString(2, cart);  
cstmt.registerOutParameter(3, Types.VARCHAR);  
cstmt.executeUpdate();  
return cstmt.getString(3);
```

ResultSet

The `executeQuery` method of `Statement`, `PreparedStatement`, and `CallableStatement` returns the `ResultSet` object as an output of the query.

The `ResultSet` object maintains a cursor to the current row of the result set of a query. This cursor can be advanced to the next row by using the `next` method of this object. The cursor by default can only be moved forward and is read-only.

The columns value for the current row can be fetched by calling the `getXXX` method of the `ResultSet` object where `XXX` denotes the data type of the column. The `getXXX` takes the column number or name for the current row as an input. Once all the rows are fetched, the `next` method returns an exception.

Example 5-4 on page 209, Example 5-5 on page 210, and Example 5-7 on page 211 use the `ResultSet` object to retrieve the result of the query.

By default, the cursor cannot be moved backward and updated. Cursors can be moved backward or moved directly to a specific row by defining the scrollability of the `ResultSet` object. Similarly, the cursor can be updated by defining the updateability of the `ResultSet` object.

The following three properties for the `ResultSet` can be set while creating the `Statement` object:

- ▶ `resultSetType`
- ▶ `resultSetConcurrency`

► resultSetHoldability

While resultSetType defines the scrollability of the cursor, resultSetConcurrency defines the updatability of the cursor. resultSetHoldability indicates that the result set will remain open even after the statement is closed. The createStatement and prepareStatement supporting all of these properties have the following syntax:

```
createStatement(int resultSetType, resultSetConcurrency,  
resultSetHoldability)  
prepareStatement(int resultSetType, resultSetConcurrency,  
resultSetHoldability);
```

All three properties are optional. Example 5-10 defines a Statement object with a scrollable and updatable ResultSet. It fetched the ResultSet row by row and update the status whenever the record with value 5003 is found. It prints all the records after the update is done.

Example 5-10 Scrollable and updatable Result set

```
Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,  
ResultSet.CONCUR_UPDATABLE);  
ResultSet rs=stmt.executeQuery("Select POID,Status from purchaseorder"  
);  
while(rs.next())  
{  
    int id=rs.getInt(1);  
    String status = rs.getString(2);  
    if(id==5003 && status.toUpperCase().compareTo("UNSHIPPED")==0)  
    {  
        System.out.println("updating status to shipped for id value "+  
                             id+".....");  
        rs.updateString(2,"Shipped");  
        rs.updateRow();  
    }  
}  
  
rs.beforeFirst();  
  
System.out.println("Printing all the purchase order record status");  
while(rs.next())  
{  
    int id=rs.getInt(1);  
    String info = rs.getString(2);  
    System.out.println("id="+id+" info="+ info );  
}
```

The supported methods for the scrollable cursor are:

- ▶ first()
- ▶ last()
- ▶ next()
- ▶ previous()
- ▶ absolute()
- ▶ relative()
- ▶ afterLast()
- ▶ beforeLast()

Tip: Each `ResultSet` object is associated with a `Statement`, `PreparedStatement`, or `CallableStatement` object. The sensitivity or the updatability cannot be defined at the `ResultSet` level. It should be defined while creating the statement object itself, which means `ResultSet` objects created for SQL statements using the same statement object will have the same properties. So once the `ResultSet` object is created, it is not possible to change the properties.

In the application `getProducts`, `isCustomer` uses the `ResultSet` object to fetch the result of the query.

BatchUpdate

With batch updates, a group of updates can be done to the database at the same time instead of updating one by one. The `Statement` object provides the following methods for batch updates:

- ▶ `addBatch` method can be used to add the SQL statements in the batch.
- ▶ `executeBatch` method will execute all the batch statements at the same time.
- ▶ `executeBatch` method will return the number of rows affected by the batch update.

Example 5-11 shows how to do the batch updates using the `Statement` object.

Example 5-11 Batch update with Statement object

```
con.setAutoCommit(false);
Statement stmt=con.createStatement();
stmt.addBatch("create table cart(pid varchar(10), quantity int");
stmt.addBatch("insert into cart values('100-100-01',4)");
stmt.addBatch("insert into cart values('100-101-01',5)");
int[] numUpdates=stmt.executeBatch();

for(int i=0;i<numUpdates.length;i++)
{
    if(numUpdates[i]==-2)
```

```

        System.out.println("Statement " + i + ": Unknown number of rows
        updated");
    else
        System.out.println("Statement " + i + ": Successful "+ numUpdates[i]
        +"rows updated");
    }

con.commit();

```

PreparedStatement and CallableStatement can also be used to make the batch updates. This can be done by running the same PreparedStatement or CallableStatement with different parameter values.

Example 5-12 gives the code snippet from the application code from the method *createCart*. The method takes an HashMap object which has the key-value pair of product name and the quantity as an input and inserts the PID and QUANTITY values using a batch update into a table. The table is also created in the same method. The name of the table is generated by appending the CID value to the word *cart*.

Example 5-12 Batch update with PreparedStatement object

```

Statement stmt=con.createStatement();

        String cart = "cart"+cid;
        String insertQuery="insert into "+cart+ " values((select pid
        from product where name=?),?)";
        String createQuery="create table "+cart+"(pid varchar(10),
        quantity int)";
        PreparedStatement pstmt=con.prepareStatement(insertQuery);
        System.out.println(insertQuery);
        stmt.executeUpdate(createQuery);
        con.setAutoCommit(false);
        Set set=hm.keySet();
        Iterator it=set.iterator();
        while(it.hasNext())
        {
            String key=(String) it.next();
            pstmt.setString(1,key);
            pstmt.setInt(2,((Integer)hm.get(key)).intValue());
            pstmt.addBatch();
            System.out.println(key +"
            "+((Integer)hm.get(key)).intValue());
        }
        pstmt.executeBatch();

```



```
con.commit();
con.setAutoCommit(true);
```

Tip: To execute a batch of statements, all the statements in a single batch should be batch compatible. *Batch compatible* statements fall in two categories:

1. Statements not having parameter markers or host expressions.
2. Different instances (with different parameter values) of the same PreparedStatement if the statement has the parameter marker.

All the statement in category 1 are batch compatible. In the same way, all the statements in category 2 are batch compatible, but none of the statements in category 1 are batch compatible with the statements in category 2.

5.4.3 MetaData

The java.sql package provides three interfaces to access the metadata at the database level, parameter level, and result set level. The driver class implements these interfaces.

DatabaseMetaData

DatabaseMetaData object gives the information of the database as a whole. The object can be created by calling the method getMetaData of Connection object. The DatabaseMetaData object provides a lot of methods to get the database information. Some of them are explained below:

- ▶ Driver information
getDriverName and getDriverVersion methods give the details of the driver in use.
- ▶ Index and primary key information
getIndexInfo gives the details of the indexes in the database. Similarly, the getPrimaryKeys method gives the primary key's details.
- ▶ Information of database object name's length
getMaxCursorNameLength, getMaxProcedureNameLength, getMaxSchemaNameLength, getMaxStatementLength, getMaxTableNameLength and so on give the maximum length allowed for the different database objects.
- ▶ Information regarding the database objects
getProcedures, getSchemas, getTables, getUsername, getTableTypes, and so on methods give the information regarding the database object.

- ▶ Information regarding database object size

Methods are provided to get the information regarding the size of the object such as maximum number of connections allowed, maximum number of columns in a table, and so on. Examples of these kinds of methods are:

- `getMaxConnection`
- `getMaxColumnInTable`
- `getMaxColumnInOrderBy`
- `getMaxColumnInIndex`
- `getMaxColumnInSelect` and so on

- ▶ Support functions

There are a lot of support functions defined that can be used to check if any of the functionality is supported in this driver or not. Some examples are:

- `supportBatchUpdates`
- `supportConvert`
- `supportColumnAliasing`
- `supportGroupBy`
- `supportGroupByUnrelated`
- `supportExpressionInOrderBy`
- `supportFullOuterJoins` and so on

ParameterMetaData

This interface is included in JDBC universal driver and is not supported for Type 2 drivers. This interface contains the methods to retrieve the information regarding the parameter's markers in a `PreparedStatement` object. The `ParameterMetaData` object can be created by calling the `getParameterMetaData` method of `PreparedStatement` object. This interface has the following methods:

- ▶ `getParameterCount`

This method returns the number of parameters in `PreparedStatement`.

- ▶ `getParameterType`

This method returns the SQL data type of the parameter marker. It takes an integer value, which indicates the position of the parameter marker in the `PreparedStatement`.

- ▶ `getParameterMode`

This method gives the information regarding the type of the parameter: `IN`, `INOUT`, or `OUT`.

- ▶ `isNullable`

This method returns `true` if the parameter is nullable; otherwise, it returns `false`.

- ▶ `getPrecision` and `getScale`

These methods return the precision and the scale of the parameter marker if the parameter type is decimal.

ResultSetMetaData

This interface provides the methods to get the `ResultSet` information. This information is specially needed to retrieve the data when the application does not know about the column types in the `ResultSet`. The object of `ResultSetMetaData` can be created by calling the `getResultSetMetaData` method on the `ResultSet` object. This interface has the following important methods:

- ▶ `getColumnCount`
This method returns the number of columns in the `ResultSet` object.
- ▶ `getColumnName`
This method returns the name of the column in the `ResultSet` object.
- ▶ `getColumnType`
This method returns the data type of the column.
- ▶ `getTableName`
This method returns the name of the underlying table for the `ResultSet`.
- ▶ `isNullable`
This method returns `true` if the `ResultSet` column can have a null value.
- ▶ `isReadOnly`
This method will return `true` if the column is read-only.

5.5 Stored procedure support

DB2 supports the Java stored procedure, which can be run on the server side by the application. Follow these guidelines to create the Java stored procedure:

- ▶ The method in the class, which will map to the stored procedure, must be defined as public static void method.
- ▶ Output and InOut parameters must be set up as a single element array.

To create, register, and use a Java stored procedure in an application, follow these steps:

1. Write a Java program with the stored procedure method. The method should be public static void.
2. Compile the program using the Java compiler.
3. Copy the `.class` file of the program to `sqllib/function` directory on server side. If you declare a class to be part of the Java package, create

subdirectories in the function directory that correspond to the fully qualified class names and place the related class files in the corresponding subdirectories. You can create a JAR file too in case you want to copy a set of stored procedure class files.

4. Register the stored procedure to the database using the CREATE PROCEDURE command.
5. Call the stored procedure from the client program.

Let us register the stored procedure to create an XML document for the porder column of the PURCHASEORDER table.

Step 1

Example 5-13 shows the stored procedure. Make sure that the class should be public and extends the StoredProc class. The method in the class should be public void. The OUT parameter values are set using the set function.

Example 5-13 Stored procedure

```
import java.sql.*;
import com.ibm.db2.jcc.*;
import COM.ibm.db2.app.StoredProc;
public class CreateOrder extends StoredProc
{
public void createOrder(int poid, String cart, String porder) throws
Exception
{
    PreparedStatement stmt;

    Connection con =
DriverManager.getConnection("jdbc:default:connection");
    java.util.Date d = new java.util.Date();
    String value=null;
    String date=d.getYear()+"-"+d.getMonth()+"-"+d.getDate();
    String query="Select xmldocument(XMLELEMENT (NAME
\"PurchaseOrder\", \" +
    \"xmlattributes(cast(? as int) as \"PoNum\",cast(? as varchar(10))
as \"OrderDate\", cast(? as varchar(10)) as \"Status\"),\" +
    \"xmlagg(xmlelement(NAME \"item\",xmlconcat(\" +
    \"xmlelement(NAME \"partid\", t.pid),\" +
    \"xmlelement(NAME \"name\", p.name),\" +
    \"xmlelement(NAME \"quantity\",t.quantity),\" +
    \"xmlelement(NAME \"price\", t.quantity*p.price)\" +
    \")))) from \" + cart +\" as t, product as p where p.pid=t.pid\";
```

```

        stmt = con.prepareStatement(query);
        stmt.setInt(1,poid);
        stmt.setString(2, date);
        stmt.setString(3,"unshipped");
        ResultSet rs=stmt.executeQuery();
        rs.next();
        value=rs.getString(1);
        rs.close();
        set(3,value);
    }
}

```

Step 2

Compile the above class using the javac compiler.

Step 3

Copy the CreateOrder.class to the sql/lib/function directory.

Step 4

Register the stored procedure using the following command:

```

CREATE PROCEDURE createOrder( IN poid int,IN cart varchar(10), OUT
porder varchar(5000)) DYNAMIC RESULT SETS 0 NOT DETERMINISTIC LANGUAGE
JAVA PARAMETER STYLE DB2GENERAL NO DBINFO FENCED THREADSAFE READS SQL
DATA PROGRAM TYPE SUB EXTERNAL NAME 'CreateOrder.createOrder'

```

Step 5

Call the stored procedure from the client program.

The createPorder method of the application calls this stored procedure to create the PORDER column value for the PURCHASEORDER table. You can call this stored procedure from the command line using the following command:

```
call createOrder(poid, cart, ?)
```

where *poid* is the purchase order ID and *cart* is the table name. Before calling the stored procedure, make sure that the *poid* value does not exist in the PURCHASEORDER table and CART table has the PID and QUANTITY columns of varchar(10) and int data type respectively. Also, populate the CART table with product ID and quantity values. PID inserted in this table should exist in the PRODUCT table.

The stored procedure returns the XML value in its output parameter denoted by the question mark (?) while calling the stored procedure.

5.6 Handling large objects

JDBC provides different classes to support BLOB, CLOB, and XML values.

BLOB

A BLOB value can be used to store the binary large value. A BLOB object can be created and populated using the way shown in Example 5-14.

Example 5-14 Creating a BLOB value

```
String data=new String("jdbc");
byte[] byteArray=data.getBytes();
java.sql.Blob
blobData=com.ibm.db2.jcc.t2zos.DB2LobFactory.createBlob(byteArray);
```

BLOB implementation in DB2 JDBC universal driver does not support any constructor function to create the BLOB object. So, it is not possible to create a BLOB object using a new operator.

The value of the BLOB type parameter can be set using the following PreparedStatement setter methods:

- ▶ setBytes
- ▶ SetBinaryStream

The BLOB value from the ResultSet object can be retrieved using the following ResultSet getter methods:

- ▶ getBytes
- ▶ getBinaryStream

CLOB and DBCLOB

A CLOB value can be used to store the character large value. A CLOB object can be created using a way similar to the way we created a BLOB object.

Example 5-15 shows how to create a CLOB value.

Example 5-15 Creating a CLOB value

```
String xsdData = new String("jdbc");
java.sql.Clob clobData =
com.ibm.db2.jcc.t2zos.DB2LobFactory.createClob(xsdData);
```

The parameter value of type CLOB can be set using the following setter methods of PreparedStatement object:

- ▶ setString

- ▶ `setAsciiStream`
- ▶ `setUnicodeStream`
- ▶ `setCharacterStream`

The CLOB value can be retrieved from the result set using the following getter methods of `ResultSet` object:

- ▶ `getAsciiStream`
- ▶ `getCharacterString`
- ▶ `getString`
- ▶ `getUnicodeStream`

DBCLOB values are handled as CLOB objects in JDBC.

Tip: When creating the BLOB object, a byte array should be passed to the `createBlob` method. When creating the CLOB object, a String value should be passed to the `createCLOB` method.

GRAPHICS and DBGRAPHICS

GRAPHICS and DBGRAPHICS are treated as String values in JDBC.

XML

XML is a new data type introduced in DB2 9. XML is different from the current data types supported in DB2, because XML is self-defined and stored in a tree structure manner in the database. For more details about the XML data type, refer to Chapter 2, “Application development with DB2 pureXML” on page 49.

Currently, JDBC does not have any XML class to support the XML data type. XML data or columns can be mapped to any of the following:

- ▶ `String`
- ▶ `byte`
- ▶ `Stream`
- ▶ `com.ibm.db2.DB2Xml` class

`com.ibm.db2.DB2Xml` is a newly introduced class in JDBC universal driver. This class is a native class for the universal driver for handling the XML values.

The following `setXXX` method can be used to set the value of the XML parameter in a `PreparedStatement` object:

- ▶ `setBytes()`
- ▶ `setString()`
- ▶ `setAsciiStream()`
- ▶ `setBinaryStream()`

- ▶ `setCharacterStream()`
- ▶ `setClob()`
- ▶ `setBlob()`
- ▶ `setObject()`
Supports `DB2Xml`, `String`, `byte[]`, `InputStream`, `Reader`, `CLOB`, and `BLOB` as parameters

The encoding conversion of the XML data depends on which `setXXX` method is used to bind the parameter value. The `setAsciiStream`, `setCharacterStream`, and `setString` methods convert the encoding to UTF-8 and send the XML value as an explicit external encoding; `setByte` and `setBinaryStream` assume that the internal encoding is correct and send the XML value with internal encoding.

An XML value can be retrieved from the `ResultSet` using the following `getXXX` methods:

- ▶ `getBytes()`
- ▶ `getString()`
- ▶ `getAsciiStream()`
- ▶ `getBinaryStream()`
- ▶ `getCharacterStream()`
- ▶ `getObject()`

The `getObject` method will retrieve the XML value as an `DB2Xml` class object. The XML values retrieved by these methods are externally encoded data and without explicit XML declaration. These methods cannot be called on the same XML row again. Trying to do that will throw an `SQLException`.

com.ibm.db2.jcc.DB2Xml Class

DB2 Universal JDBC driver introduces the `DB2Xml` class. The object of this class is returned to the application whenever the `getObject` method is used to retrieve the XML column value from the `ResultSet` object.

This class defines methods to convert the XML data to other Java data types, such as `String`, `byte[]`, `BinaryStream`, and `AsciiStream`. These methods have the `getDB2XXX` format where `XXX` indicate the data type. This class also defines the methods which convert the XML data to the specified data and add the XML declaration tag. These methods have a format such as `getDB2XmlXXX`.

Example 5-16 shows how to retrieve an XML value as a `DB2Xml` object.

Example 5-16 Retrieving XML value as a DB2Xml object

```
Statement stmt=con.createStatement();  
ResultSet rs=stmt.executeQuery("select info from customer");
```



```
while(rs.next())
{
    com.ibm.db2.jcc.DB2Xml data=(com.ibm.db2.jcc.DB2Xml)
    rs.getObject(1);
    // Print the result as DB2 XML String
    System.out.println();
    System.out.println(data.getDB2String());
}
// Close the result set
rs.close();
```

The metadata information regarding the XML data type column gives the data type of the column as `java.sql.Types.OTHER` because there is no XML data type defined in the JDBC standard.

5.7 Simple application program life cycle

Figure 5-1 on page 226 shows different life cycles for a query in an application program. An application program starts by getting the Connection object for the database using DriverManager. A Connection object can be used to create three different kinds of SQL statements: Statement, PreparedStatement, and CallableStatements. Connection objects provide methods to create any of these objects. A Statement object is used to execute the SQL statement with a parameter, and the PreparedStatement can be used to run the SQL statement with a parameter. CallableStatement is used to call the user defined functions (UDFs). Figure 5-1 depicts different paths for an SQL statement in a JDBC program.

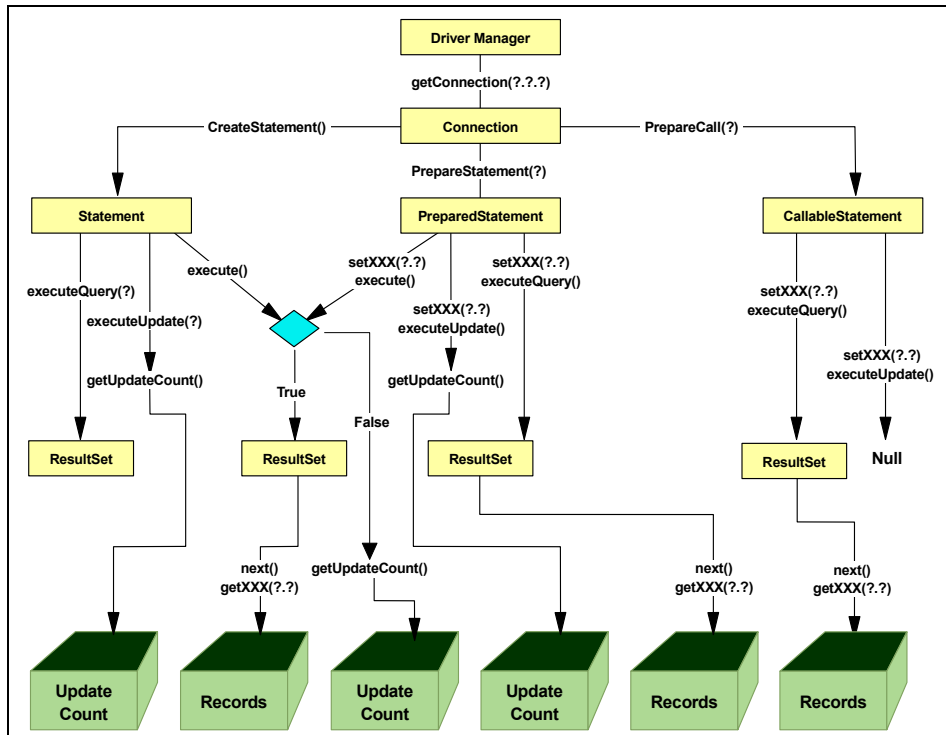


Figure 5-1 Different path for an SQL statement in a JDBC program

Each of the three classes, Statement, PreparedStatement, and CallableStatement, provides three different methods to execute a query:

- ▶ `executeQuery`: This method is used to select the rows and return a `ResultSet` object.
- ▶ `executeUpdate`: This method is used to update the database and return the update count.
- ▶ `execute` method: This method returns a Boolean value of `true` if the output of the query is a `ResultSet` object (`SELECT`) and `false` if its an update statement (`INSERT`, `UPDATE`, or `DELETE`).

The `getUpdateCount` method can be used to get the number of rows updated in the database by the SQL statement. The `getXXX` method of the `ResultSet` object can be used to fetch the individual rows.

For `CallableStatement`, each of these methods has a slightly different meaning. The `executeQuery` method is used when the stored procedure returns a `ResultSet`, where `execute` is used when the stored procedure returns multiple `ResultSets`.

5.8 Introduction to javax.sql package

The javax.sql package defines extension to the JDBC 2.0 API to support connection pooling, data source, distributed transaction, and so on. DB2 universal database implements all of these interfaces. DataSource implementation is explained below.

5.8.1 DataSource

Using DriverManager to get the database connection makes the application dependent on the vendor specific URL and driver classes. A DataSource object can be used to get the connection, which is portable among the different data sources, hence, not dependent on the specific URL format and driver classes. This portability is gained by registering the different data sources with the JNDI service (javax.naming service) and getting the information about any data sources by providing the logical name of the datasource in the application. A DataSource object is either created in the application itself or by a lookup in the JNDI service. Before making use of lookup in JNDI services, you need to register the database to JNDI.

The Example 5-17 gives an example of creating DataSource object by JNDI lookup service and getting the connection to the data source.

Example 5-17 Creating DataSource by JNDI lookup service

```
Context ctx=new InitialContext();
DataSource ds=(DataSource)ctx.lookup("sampledb");
Connection con=ds.getConnection();
```

A DataSource object can be bound to a logical name by registering it to JNDI. Registering to JNDI services requires setting all of the properties to get a connection such as database name, user name, password, and so on. Once the data source is registered, an application needs to know only the logical name to retrieve the database connection. DB2 universal driver provides the following implementation for the DataSource:

```
com.ibm.db2.jcc.DB2SimpleDataSource
```

A data source can be registered using this implementation as shown in Example 5-18.

```
DB2SimpleDataSource db2ds=new com.ibm.db2.jcc.DB2SimpleDataSource();
db2ds.setDatabaseName("sample");
db2ds.setDescription("sample database for DB2 Express-C");
Context ctx=new InitialContext();
ctx.bind("sampledb",db2ds);
```

5.9 Exception handling

JDBC exception handling is done using the try-catch block of the Java application. A DB2 application throws an `SQLException` whenever it encounters the SQL error while running the SQL statements.

5.9.1 `SQLExceptions`

An object of `SQLException` is created and thrown whenever an error occurs while accessing the database. The object gives the following information regarding the error:

- ▶ **Message:**
Message is a textual representation of the error code. The `getMessage` method of the `SQLException` object returns this message string.
- ▶ **SQLState:**
The `SQLState` string can be retrieved from the `SQLException` object by calling the `SQLState` method.
- ▶ **ErrorCode:**
This is an integer value and indicates the error which caused the exception to be thrown. Error codes can be retrieved by calling the `getErrorCode` method of the `SQLException` object.

Apart from the above information, DB2 JCC driver provides an extra interface `com.ibm.db2.jcc.DB2Diagnosable`. This interface gives more information regarding the error that occurred while accessing the DB2 database. `DB2Diagnosable` interface has the following methods:

- ▶ `getSqlca`
This method returns the `db2sqlca` object, which gives an SQL error code, `SQLERRMC` values, `SQLERRP` value, `SQLERRD` values, `SQLWARN` values, and `SQLState` value.
- ▶ `getThrowable`

This method returns the `java.lang.Throwable` object that causes the `SQLException` if one exists.

► `PrintTrace`

This method prints the stack trace information.

If multiple `SQLExceptions` are thrown, they are chained. The next exception information can be retrieved by calling the `getNextException` method of the current `SQLException` object. This method will return null if the current `SQLException` object is last in the chain. A while loop in the catch block of the program can be used to retrieve all the `SQLException` objects one by one.

Example 5-19 shows how to handle the `SQLException` in the try-catch block.

Example 5-19 SQLException handling

```
try {
    // code which can throw SQLException go here
} catch (SQLException sqle)
{
    if(sqle instanceof DB2Diagnosable)
    {
        com.ibm.db2.jcc.DB2Diagnosable diag =
            (com.ibm.db2.jcc.DB2Diagnosable) sqle;
        DB2Sqlca sqlca = diag.getSqlca();
        if(sqlca != null)
        {
            int sqlCode=sqlca.getSqlCode();
            String sqlErrmc = sqlca.getSqlErrmc();
            String[] sqlErrmcTokens = sqlca.getSqlErrmcTokens();
            String sqlErrp = sqlca.getSqlErrp();
            // Get the SQLERRP
            int[] sqlErrd = sqlca.getSqlErrd();

            // Get SQLERRD fields
            char[] sqlWarn = sqlca.getSqlWarn();

            // Get SQLWARN fields
            String sqlState = sqlca.getSqlState();

            // Get SQLSTATE
            System.err.println
                ("----- SQLCA -----");
            System.err.println ("Error code: " + sqlCode);
            System.err.println ("SQLERRMC: " + sqlErrmc);
            for (int i=0; i< sqlErrmcTokens.length; i++) {
```

```

        System.err.println (" token " + i + ": "
            + sqlErrmcTokens[i]);
    }
    System.err.println ( "SQLERRP: " + sqlErrp );
    System.err.println (
        "SQLERRD(1): " + sqlErrd[0] + "\n" +
        "SQLERRD(2): " + sqlErrd[1] + "\n" +
        "SQLERRD(3): " + sqlErrd[2] + "\n" +
        "SQLERRD(4): " + sqlErrd[3] + "\n" +
        "SQLERRD(5): " + sqlErrd[4] + "\n" +
        "SQLERRD(6): " + sqlErrd[5] );
    System.err.println (
        "SQLWARN1: " + sqlWarn[0] + "\n" +
        "SQLWARN2: " + sqlWarn[1] + "\n" +
        "SQLWARN3: " + sqlWarn[2] + "\n" +
        "SQLWARN4: " + sqlWarn[3] + "\n" +
        "SQLWARN5: " + sqlWarn[4] + "\n" +
        "SQLWARN6: " + sqlWarn[5] + "\n" +
        "SQLWARN7: " + sqlWarn[6] + "\n" +
        "SQLWARN8: " + sqlWarn[7] + "\n" +
        "SQLWARN9: " + sqlWarn[8] + "\n" +
        "SQLWARNA: " + sqlWarn[9] );
    System.err.println ("SQLSTATE: " + sqlState);
    }
else
    {
        System.out.println(sqlc.getMessage());
        System.out.println(sqlc.getErrorCode());
        sqlc.printStackTrace();
    }
    System.out.println("Rollback the transaction and quit the
        program");
    System.out.println();
    try { con.rollback(); }
    catch (Exception e) {}
    System.exit(1);
}

```

Note: Before using the method of `Diagnosable` class, make sure that the object of the `SQLException` thrown is an instance of this class. If not, you cannot use the method defined in this class. In that case, use the standard `getMessage`, `getSQLState`, and `getErrorCode` methods of `SQLException` to print the information.

The `SQLException` class has multiple subclasses, which define more specific errors. We explain these classes next.

5.9.2 SQLWarning

The `SQLWarning` object is created whenever there is a database warning that occurred while calling the methods of the following classes:

- ▶ `Connection`
- ▶ `Statement`
- ▶ `PreparedStatement`
- ▶ `CallableStatement`
- ▶ `ResultSet`

All these interfaces contain the `getWarning` method to retrieve the warning information. Note that the creation `SQLWarning` object does not throw any `SQLException`. You need to call the `getWarning` method of the above interface to check if any warning exists or not. See Example 5-20.

Example 5-20 Handling SQL warning

```
Statement stmt=con.createStatement();
stmt.executeUpdate("delete from product where pid='101'");
SQLWarning sqlwarn=stmt.getWarnings();
while(sqlwarn!=null)
{
    System.out.println ("Warning description: " + sqlwarn.getMessage());
    System.out.println ("SQLSTATE: " + sqlwarn.getSQLState());
    System.out.println ("Error code: " + sqlwarn.getErrorCode());
    sqlwarn=sqlwarn.getNextWarning();
}
```

5.9.3 DataTruncation

The `DataTruncation` class is a subclass of the `SQLWarning` to handle the data truncation whenever an application tries to insert a value larger than the value defined in the database definition. In that case, the data is stored after truncating

the value to the specified size and a `DataTruncation` object is created to provide the information. Whenever truncation happens while reading the value from the database, an `SQLWarning` object is created instead of `DataTruncation`. A `DataTruncation` object is created when truncation occurred during writing to the database.

All the methods in `SQLException` and `SQLWarning` classes are inherited in this class. Apart from these, the following methods give more information regarding `DataTruncation` Exception:

- ▶ `getParameter`
This method returns `true` if the parameter value is truncated. It returns `false` if the column value is truncated.
- ▶ `getIndex`
This method will return an integer value, which gives the index of the parameter or column being truncated. This method will return `-1` if the index of the column or parameter is unknown.
- ▶ `getRead`
This method will return `true` if the truncation occurs while reading. `false` is returned if truncation occurred while writing to the database.
- ▶ `getDataSize`
This methods returns the actual size of the data, which should be read or written to the database without truncation.
- ▶ `getTransferSize`
This method will return the size of the data, which is actually read or written to the database with truncation.

5.9.4 BatchUpdateException

An object of `BatchUpdateException` is thrown whenever an error occurs while running a set of statements together by using `BatchUpdate`. This class inherits all the methods from `java.lang.Exception` class. Apart from that, the following method is provided by the `BatchUpdateException` for the additional information:

- ▶ `getUpdateCounts`:
This method returns a array whose size is equal to the number of elements in the batch. This array has an entry for each statement in the batch to indicate the failure or success of the statement.

Example 5-21 on page 233 shows the catch block of the method `createCart` from our application example. This method does the batch update to update the cart table.


```
catch (BatchUpdateException buex) {
    int [] updateCounts = buex.getUpdateCounts();
    for (int i = 0; i < updateCounts.length; i++) {
        System.err.println(" Statement " + i + ":" +
updateCounts[i]);
        System.err.println(" Message: " + buex.getMessage());
        System.err.println(" SQLSTATE: " + buex.getSQLState());
        System.err.println(" Error code: " + buex.getErrorCode());
        SQLException ex = buex.getNextException();
        while (ex != null) {
            System.err.println("SQL exception:");
            System.err.println(" Message: " + ex.getMessage());
            System.err.println(" SQLSTATE: " + ex.getSQLState());
            System.err.println(" Error code: " + ex.getErrorCode());
            ex = ex.getNextException();
        }
    }
}
```

5.10 Transactions

Transaction are used to access data concurrently with consistency and by maintaining data integrity. You can view a database transaction as a set of interactions with the database system independent of other transactions, which are either completed or aborted. This set of interactions is called *unit of work*. A unit of work can be defined as a set of SQL statements, which need to be executed successfully to complete a task. Any failure from these SQL statements should lead to failure of the transaction (unit of work), and the database system should be restored to the state where it is before starting the transaction.

A single transaction can contain a set of queries, which reads and writes to the database. To complete a transaction, an application should make sure that all of the queries are executed successfully before ending the transaction. This is required to maintain the integrity of the data. For example, for an order, if the customer bought x quantity of the product y , the inventory details for product y should be decreased by the x amount to complete the purchase order transaction. If the purchase order is successful without updating the inventory, the data in the inventory details becomes inconsistent.

A typical transaction has the following steps:

1. Start the transaction.
2. Execute some queries. (Any updates to the database are not visible to the outside world yet.)
3. Complete the transaction by committing. (Any updates to the database become visible now.)

To maintain the consistency and integrity in a transaction, JDBC includes the following concepts:

- ▶ Auto commit mode
- ▶ Transaction isolation level
- ▶ Savepoints

5.10.1 Auto commit mode

A transaction is started whenever an SQL statement requires one and there is no transaction in place. There is no explicit API defined to start a transaction.

Auto commit mode is set to indicate when to end a transaction. Auto commit mode can be set by setting the auto-commit attribute of the Connection object. This attribute is set to either enable or disable. Enabling this attribute makes each SQL statement a separate transaction. So whenever an SQL statement is issued by the application, a new transaction is started and once the statement is executed successfully, the transaction is completed by committing any updates done to the database. Disabling this attribute gives the application flexibility to end the transaction any time by calling the commit method of the Connection object. If any of the SQL statements in the transaction failed, the application needs to restore the state of the database by calling the rollback method of the Connection object. Disabling this attribute gives the flexibility to the application to include multiple SQL statements in a transaction and commit and roll back the transaction whenever needed.

The value of the auto-commit attribute of the Connection object can be set by calling the Connection object method `setAutoCommit`. The default value for this attribute is `true` (enable). If the value of the auto-commit attribute is changed in the middle of the transaction, the current transaction is committed and a new transaction with the changed value is started.

5.10.2 Transaction isolation level

The isolation level specifies how the data is visible to the transactions running concurrently while the transactions update the data in the database. Isolation level also relates to how the data is locked for a particular transaction. They

directly impact the level of concurrent access to the same database object and how the different transactions interact with each other while accessing the same data source. The different kinds of interaction possibilities are:

▶ **Lost updates**

Two applications, Application A and Application B, read the same row and calculate the new value for a column for that row. Application A updates the row with the new value, and just after that Application B updates its new value. The value updated by Application A is lost.

▶ **Access to uncommitted data**

Application A updates a row without committing, and Application B reads this uncommitted value. Now, if Application A rolled back the transaction, Application B has already read the data and has the wrong set of data.

▶ **Nonrepeatable reads**

Application A reads from the database and then goes on to process another SQL statement in the transaction. In the meantime, Application B has updated or deleted the same row. Now if Application A came back and read the same data again, it found either the value is updated or deleted from the database. This type of interaction is possible when Application B updates or deletes the row, which was the part of the result set of transaction A.

▶ **Phantom read**

Application A selects some data based on some condition. Application B inserts another row to the same table, which satisfies the condition of Application A and commits the changes. If Application A selects the data again, it will find the extra rows in the result of the query. This kind of transaction is possible if Application B adds some more rows to the table, which can be part of the result set of transaction A if the result set is created again (by running the statement again).

These interactions between different transactions can cause unpredictable results. Setting isolation levels to appropriate values restricts these interactions. DB2 provides the following isolation levels. Each isolation level is described on two parameters:

- ▶ How the data is seen by the other transactions while this transaction with this specified isolation level is updating or reading the data.
- ▶ How the current transaction with the specified isolation level can see the data read or updated by other transactions.

All the transactions acquire an exclusive lock whenever they update a row in the table, so the lost update is not possible at all.

Uncommitted read (UR)

This is the lowest level of isolation:

- ▶ Any row already read by the current transaction can be updated by another transaction.
- ▶ The current transaction can see any uncommitted data from other transactions.

Any kind of interaction except lost update is possible with this isolation level.

Cursor stability (CS)

The characteristics of CS are:

- ▶ Any row except the current row read by this transaction can be updated by the other transactions.
- ▶ The current transaction cannot see the uncommitted data from other transactions.
- ▶ Non-Repeatable read and phantom read are possible with this isolation level.

Read stability (RS)

The characteristics of RS are:

- ▶ Any row read by the current transaction cannot be updated by any other transaction.
- ▶ The current transaction cannot see the uncommitted data from other transactions.
- ▶ Interactions of type phantom read are possible between concurrent transactions.

Repeatable read (RR)

This is the highest level of isolation. This isolation level completely isolates the concurrent transactions:

- ▶ Any row read by the current transaction cannot be updated by any other transaction. At the same time, this isolation level makes sure that the ResultSet of the same statement remains constant even if the statement is executed twice in the same transaction.
- ▶ Current transactions cannot see the uncommitted data from other transactions.
- ▶ The isolation level for a transaction can be set by calling the method `setTransactionIsolation` of Connection object.

Example 5-22 on page 237 shows how to set the isolation level for a transaction.

Example 5-22 Setting isolation level

```
Connection con = null;  
Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();  
con = DriverManager.getConnection("jdbc:db2:sample");  
// Set the isolation level to RR  
con.setTransactionIsolation(TRANSACTION_SERIALIZABLE);
```

Table 5-1 show the JDBC equivalent variable for the different isolation levels.

Table 5-1 Equivalent JDBC and DB2 isolation levels

JDBC integer constant value	DB2 isolation level	Equivalent integer value
TRANSACTION_SERIALIZABLE	Repeatable read	8
TRANSACTION_REPEATABLE_READ	Read stability	4
TRANSACTION_READ_COMMITTED	Cursor stability	2
TRANSACTION_READ_UNCOMMITTED	Uncommitted read	1

5.10.3 Savepoints

A *savepoint* defines a particular state of the database during a unit of work or transaction. A savepoint is required if we want to roll back the transaction to a particular state instead of rolling back to the start of the transaction. DB2 Universal driver provides the method `setSavepoint` of the `Connection` object to set a savepoint during the transaction.

A savepoint can be release by calling the method `releaseSavePoint` method of the `Connection` object. Releasing a savepoint will release all the savepoints created after the released savepoint. After releasing the savepoint, the transaction cannot be rolled back to the released savepoint or any of the savepoints created subsequently.

A transaction can be rolled back to a savepoint by calling the `rollback` method and giving the savepoint variable as an argument to this method.

Any cursor opened inside a savepoint will remain open even after rollback to the savepoint but can go to an invalid state if they depend on some DDL statements which were rolled back.

Example 5-23 on page 238 shows how to create the savepoint.

Example 5-23 Savepoint

```
con.setAutoCommit(false);
Statement stmt=con.createStatement();
stmt.executeUpdate("create table order(id int, description
varchar(100))");
con.commit();
stmt.executeUpdate("insert into order values(1, 'first order of the
day')");
Savepoint svpt1=con.setSavepoint();
stmt.executeUpdate("insert into order values(2, 'second order')");
Savepoint svpt2=con.setSavepoint();
stmt.executeUpdate("insert into order values(3, 'third order')");
con.rollback(svpt2);
con.releaseSavepoint(svpt1);
stmt.close();
con.commit();
```

5.11 SQL/XML and XQuery support

An XQuery or SQL/XML statement can be executed in the same manner as we execute an SQL statement in the JDBC application. All the restrictions of XQuery are also applicable here. For the XQuery statement, the query should be prefixed with the `xquery` word.

For more details about how to write an XQuery, refer to Chapter 2, “Application development with DB2 pureXML” on page 49.

Example 5-24 shows how to run an XQuery statement in Java. This XQuery gives all the customers’ cities in Canada.

Example 5-24 Running XQuery

```
Statement stmt = con.createStatement();
String query="XQUERY"+
+" declare default element namespace \"http://posample.org\";"+
"fn:distinct-values(db2-fn:xmlcolumn('CUSTOMER.INFO')"+
"/customerinfo/addr[@country=\"Canada\"]/city)";
System.out.println();
System.out.println(query);
ResultSet rs = stmt.executeQuery(query);
// retrieve and display the result from the query
while (rs.next())
{
```

```
// retrieve the data as binary stream
String data=rs.getString(1);
// Print the result
System.out.println(data);
}
// Close the resultset
rs.close();
// Close the statement
stmt.close();
```

5.12 SQLj support

JDBC runs all the SQL statements dynamically, which means it prepares all the statements at runtime just before the execution. Some of the statements (for example, statements without parameters) can be prepared at compile time, which increases the performance of the application at runtime. SQLj gives this flexibility of preparing the statements at compile time. An SQLj program runs all the statements statically. The SQL statements can be embedded into the Java program using SQLj. Example 5-25 gives the different syntaxes of how to embed a static SQL statement in a Java program.

Example 5-25 SQLj syntax

```
#sql [connection-context] { sql statement }
#sql [connection-context, execution context] { sql statement }
#sql { sql statement }
#sql [execution context] { sql statement }
```

Connection context in SQLj is equivalent to the Connection object in JDBC and required for executing any SQL statement. Execution context is required to get the information regarding the SQL statement before and after executing the SQL statement. A default connection context is used when no connection context is specified.

The SQLj program needs to be translated and customized first before compiling and running the program. Translation and customization create the package for embedded SQL statements in the program and store it in the database. These packages are used by the application at runtime.

5.12.1 Getting connection context

The SQLj program needs to connect to the database before executing any SQL statement. For getting a connection context, following these steps:

1. Declare a class for connection context. This can be done using the following statement:

```
#sql context context-class-name
```

This declaration creates a class with the name `context-class-name`. To use this class to create a context object, declaration must be global and should not be inside the class.

2. Load the JDBC driver. You do this the same way we have done for the JDBC program.
3. Invoke the constructor of the context class.

The Example 5-26 shows how to establish the connection in the SQLj program.

Example 5-26 SQLj connection context

```
#sql context ctx; // This should be outside the class
Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
ctx ctx1=new ctx("jdbc:db2:sample",false);
```

A Connection context for the SQLj program can also be created using the Connection object. See Example 5-27.

Example 5-27 SQLj Connection Context from Connection object

```
#sql context ctx; // This should be outside the class
Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
Connection con=DriverManager.getConnection();
ctx ctx1=new ctx(con);
```

5.12.2 Manipulating data

A connection context class object can be used to execute any SQL statement. To monitor and control the SQL statements while executing, we need to create the ExecutionContext class. An ExecutionContext class object can be created by calling the method `getExecutionContext` of connection context. Some of the ExecutionContext methods are applicable before executing the SQL statement while some of them are applicable only after execution of the statement.

Example 5-28 shows how to create the `ExecutionContext` object and use its method `getUpdateCount` to retrieve the number of rows deleted by the `DELETE` statement.

Example 5-28 Creating ExecutionContext object

```
#sql context ctx; // this should be outside the class
String url = "jdbc:db2:sample";
Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
Connection con=DriverManager.getConnection(url);
ctx ctx1=new ctx(con);
ExecutionContext exectx1=ctx1.getExecutionContext();
#sql [ctx1,exectx1] { DELETE FROM purchaseorder WHERE
status='UnShipped' }
int i=exectx1.getUpdateCount();
```

5.12.3 Iterators

SQLj defines *iterators* for selecting multiple rows using the `SELECT` statement. Iterators are the SQLj equivalent of `ResultSet` in Java and cursors in other programming languages.

To iterators, the application needs to create an iterator class by defining the iterator and an object of that class. The result of the `SELECT` statement can be assigned to this object.

SQLj provides two types of iterators:

- ▶ Named iterators
- ▶ Positioned iterators

Named iterators

Named iterators identify a row by the name of the column in the result set. So while defining the named iterator, you need to specify the name of the columns and their data types selected by the `SELECT` statement.

Example 5-29 shows how to use the named iterator in an SQLj program.

Example 5-29 Using named iterators

```
#sql iterator namediterator(int poid,String status)
namediterator iterator1;
#sql [ctx1] iterator1={ select poid,status from purchaseorder };
while(iterator1.next())
{
    System.out.println("poid: " + iterator1.poid() + "Status: "+
```

```
        iterator1.status());
    }iterator1.close();
```

Positioned iterators

A *positioned iterator* identifies a row by its position in the result set. So while defining the position iterator, you need to give only the data types of the columns.

Example 5-30 shows how to use the positioned iterator in SQLj.

Example 5-30 Using positioned iterators

```
#sql iterator positionedIterator(int, String);
String status=null;
int poid=0;
positionedIterator iterator1;
#sql [ctx1] iterator1={ select poid,status from purchaseorder };

#sql { fetch :iterator1 into :poid, :status };
while(!iterator1.endFetch())
{
    System.out.println("poid: " + poid + "Status: "+ status);
    #sql { fetch :iterator1 into :poid, :status };
}
```

Updatable and scrollable iterators

Like JDBC, by default iterators in SQLj are read-only and cannot be moved backward. To define a scrollable iterator, you need to implement the `sqlj.runtime.Scrollable` while defining the iterator. Similar to defining the updatable cursor, you need to implement `sqlj.runtime.ForUpdate` while defining the iterator. Unlike JDBC, when defining the updatable iterator, you also need to specify the columns you would like to update. Example 5-31 gives the code snippet, which uses updatable iterators.

Example 5-31 Updatable iterator

```
#sql public iterator namediterator implements sqlj.runtime.ForUpdate
with (updateColumns="STATUS") (int poid, String status);
namediterator iterator1;
#sql [ctx1] iterator1={ select poid,status from purchaseorder };

while(iterator1.next())
{
    System.out.println("before update poid: " + iterator1.poid() +
        "Status: "+ iterator1.status());
}
```

```
    if(iterator1.status().toUpperCase().compareTo("UNSHIPPED")==0)
        #sql [ctx1] {update purchaseorder set status=
                        'shipped' where current of :iterator1 };
    }
#sql [ctx1] {commit};
```

Similarly, you can define the holdability of the iterator by adding the clause with (holdability=true) while defining the iterator.

5.12.4 Batch updates with SQLj

SQLj supports batch updates the same way JDBC does. But unlike JDBC, SQLj allows you to add statements of different types (a different instance of the same statement, or a statement with a host expression) in the same batch.

To create a batch in SQLj, an ExecutionContext is required. The batching can be enabled by calling setBatching method of ExecutionContext and setting the value to true.

The number of statements in the batch can be limited to a value by calling the method setBatchLimit of the ExecutionContext. Once this limit is set, batch will be automatically executed when this limit is reached. Batch can be explicitly executed by calling the executeBatch method of the ExecutionContext. Apart from that, a batch is executed implicitly if the batch contains a statement, which is incompatible with other statements in the same batch. In that case, a batch will be executed and a new batch is created for incompatible statements.

Example 5-32 shows how to perform batch update.

Example 5-32 Batch updates

```
ExecutionContext ec=ctx1.getExecutionContext();
ec.setBatching(true);
#sql[ctx1] { insert into product(pid, price) values('100-201-03',10) };
#sql[ctx1] {update purchaseorder set status='shipped' where poid=5000};
#sql[ctx1] {Delete from purchaseorder where poid=5010};
ec.executeBatch();
System.out.println("Batch executed");
ec.setBatching(false);
```

When an error occurs while executing any of the batch statements, the remaining statements are executed and a BatchUpdateException is thrown after all of the statements have executed.

5.12.5 Savepoints

A savepoint can be created in an SQLj program using universal driver. A savepoint in an SQLj program can be created using the SAVEPOINT statement.

Example 5-33 shows how to create a savepoint in an SQLj program.

Example 5-33 Creating a savepoint in SQLj

```
con.setAutoCommit(false);
ctx ctx1=new ctx(con);
#sql[ctx1] {create table order(id int, description varchar(100)) };
#sql[ctx1] {insert into order values(1, 'first order of the day')};
#sql[ctx1] {SAVEPOINT SVPT1 ON ROLLBACK RETAIN CURSORS};
#sql[ctx1] {insert into order values(2, 'second order')};
#sql[ctx1] {SAVEPOINT SVPT2 ON ROLLBACK RETAIN CURSORS};
#sql[ctx1] {insert into order values(3, 'third order')};
#sql[ctx1] {ROLLBACK TO SAVEPOINT SVPT2};
#sql[ctx1] {RELEASE SAVEPOINT SVPT1};
#sql[ctx1] {commit};
```

5.12.6 XQuery and SQL/XML support

An XQuery can only be run dynamically. As SQLj runs every query statically, to run the XQuery in SQLj, we need to change the XQuery statement into an SQL/XML statement. An SQL/XML statement can be run statically. For example, the SQL/XML equivalent of query in Example 2-11 on page 71 is shown in Example 5-34.

Example 5-34 SQL/XML equivalent of XQuery statement

```
select xmlquery('$d/movie[movie-details/country="US"]/heading/title'
passing movies.info as "d") from movies
```

Example 5-35 shows the SQLj code to run the query shown in Example 5-34.

Example 5-35 Running XQuery in SQLj

```
#sql custIter = {select
xmlquery('$d/movie[movie-details/country="US"]/heading/title' passing
movies.info as "d") from movies};
```

For retrieving XML values, an iterator can be defined either with Object data type argument or a new class, com.ibm.db2.jcc.DB2Xml object.

5.12.7 Exception handling

Exception handling can be done in the same way we handle exceptions in a JDBC program.

5.12.8 JDBC and SQLj

JDBC and SQLj can be used together in a single application. A connection object in JDBC is similar to the `ConnectionContext` in SQLj. A connection object can be retrieved from a `ConnectionContext` object and vice versa. Getting a `ConnectionContext` object from a `Connection` object is shown in Example 5-27 on page 240. A connection object from `ConnectionContext` can be retrieved using the method `getConnection` of `ConnectionContext` object.

In the same way, an iterator in SQLj and the JDBC `ResultSet` can be retrieved from each other. To get an iterator from the `ResultSet` object, use the following command:

```
#sql iterator={CAST :result-set }
```

Before doing the cast, make sure that iterator definition is the same as the result set definition. For the named iterator, column name and the data types of the columns defined in the iterator should be the same as those of the result set. For a positioned iterator, the number of columns and data type should match those of the result set. Apart from that, properties such as scrollability, updatability, and holdability should match the definition of the iterator. For details about setting these properties for SQLj iterator, refer to “Updatable and scrollable iterators” on page 242.

A `ResultSet` object can be retrieved from the iterator by calling the `getResultSet` method of the iterator class.

Example 5-36 shows how to get an iterator from the `ResultSet` object.

Example 5-36 Creating an iterator from the ResultSet object

```
#sql public iterator positionIterator (int, String);

Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
Connection con=DriverManager.getConnection(url);
con.setAutoCommit(false);
ctx ctx1=new ctx(con);
positionIterator iterator;
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select poid, status from
purchaseorder");
```

```

#sql [ctx1] iterator={cast :rs};
#sql {fetch :iterator into :poid, :status};
while(!iterator.endFetch())
{
    System.out.println("id: "+poid+" status: "+status);
    #sql {fetch :iterator into :poid, :status};
}
iterator.close();

```

Similarly, an ExecutionContext in SQLj is equivalent to the Statement object in JDBC. In SQLj, ExecutionContext is used to get the information regarding the Query statement before and after execution; in a Java Statement, the object is used to do the same. Some of the equivalent functions are shown in Table 5-2.

Table 5-2 Comparison between Statement and ExecutionContext

Method description	Statement object method	ExecutionContext method
Retrieve the update counts done by the update or delete statement	getUpdateCount	getUpdateCount
Retrieve the warnings thrown by the statements	getWarnings	getWarnings
To start the batchUpdate	no need to enable batching.	setBatching
To add the SQL statement in the batch	addBatch	After enabling the batching using setBatching method, Subsequent statements are added to the batch automatically
To execute the batch statements	executeBatch	executeBatch

Table 5-3 on page 247 gives you a comparison between ResultSet object in JDBC and iterator in SQLJ.

Table 5-3 Comparison between ResultSet and iterator

Description	ResultSet object	Iterator
Creation	Returned by Statement object methods executeQuery. For example, <code>ResultSet rs=null; rs=stmt.executeQuery(query)</code> where <code>stmt</code> is the statement object.	Need to declare the iterator class using SQLj declaration clause. Create the object after declaration. For example: <code>#sql iterator iterator1(int, String); iterator iterator1</code>
Scrollability and Updatability	Scrollability and Updatability are defined while creating the Statement object for all the ResultSets created from that object.	Scrollability and Updatability are defined by specifying the <code>implement</code> clause in the corresponding interface while declaring the iterator.
Holdability	Same as above.	Holdability is defined by defining the <code>holdability</code> property to <code>true</code> in the <code>with</code> clause of the iterator.
Methods supported for scrollability	first last previous next absolute relative afterLast beforeFirst	first last previous next absolute relative afterLast beforeFirst

5.13 Running the application

The application code in Example 5-1 on page 202 can be executed either stand-alone or using a Web service.

5.13.1 Running an application stand-alone

To run the application stand-alone, follow these steps:

1. Set your CLASSPATH to include all the files required for JDBC (see “Application requirements” on page 200).
2. Initialize the following variables in the `main` method:
 - a. `custid`: Initialize the variable with a valid customer ID (Exists in customer table).

- b. hm: This variable is of type HashMap. The key for the HashMap is product name and value is the quantity ordered.
3. Start the DB2 server.
4. Create the SAMPLE database using this command:

```
db2samp1 -xml
```
5. Register the procedure created in “Stored procedure support” on page 219 to the database.
6. Compile the Java code using the Java compiler.
7. Run the application.

5.13.2 Running the application as a Web service

To run the application as a Web service, follow these steps:

1. Install the Tomcat application server. You can obtain Tomcat from the following Web site:
<http://tomcat.apache.org/download-55.cgi>
2. Install the Eclipse SDK.
3. Install the Tomcat-eclipse plug-in.
4. Download the application code from the IBM Redbooks Web site. For download instructions, refer to Appendix C, “Additional material” on page 319.
5. Open the project in eclipse.
6. Start the DB2 server.
7. Create the SAMPLE database using the command:

```
db2samp1 -xml
```
8. Register the procedure created in “Stored procedure support” on page 219 to the sample database.
9. Start the Tomcat server.
10. Point your browser to this link:

```
http://localhost:8080/cart/default.jsp
```

If Tomcat and the application are downloaded onto a remote machine, give the machine IP address/name instead of *localhost*.

Note: The previous project can be run using any application server. We recommend Tomcat and Eclipse, because we used them to create the application.



Application development with .NET

In this chapter, we introduce DB2 application development using .NET.

In this chapter, we discuss following:

- ▶ Requirements for .NET application development with DB2
- ▶ Add-in features for Visual Studio .NET
- ▶ DB2 Data Providers available for use with .NET
- ▶ Application examples using .NET and DB2 Express-C

6.1 .NET technology and ADO.NET

.NET is Microsoft's Web services architecture, which represents a set of Microsoft frameworks and technologies. It is alternative to J2EE, the distributed application infrastructure based on Java language and is available for Windows platforms.

ActiveX Data Object for .NET (ADO.NET) provides classes, methods, and attributes to access data source utilizing features of Web services and XML.

ADO.NET serves as a single data access layer used by all server processes and applications running on Microsoft platforms. ADO.NET consists of objects such as Connection, Command, DataAdapter, and DataReader to access data source as shown in Figure 6-1.

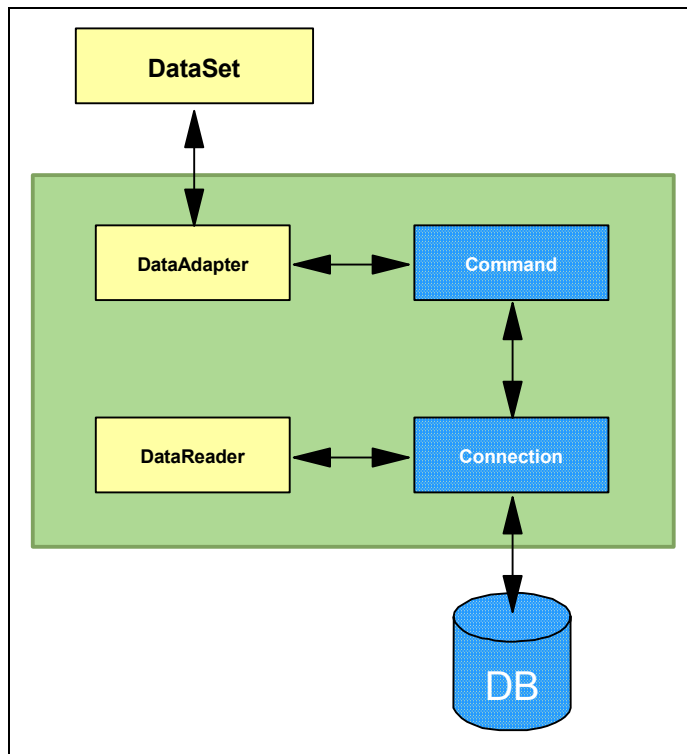


Figure 6-1 ADO.NET architecture

6.2 Requirements for .NET application development with DB2

The following requirements are needed to develop .NET application solutions with DB2 data source:

- ▶ DB2 client (minimum DB2 Application Development Client) Version 8.1.2 for use of DB2 .NET Data Provider and Version 8.1.5 for OLE DB bridge provider needs to be installed and the appropriate database needs to be cataloged.
- ▶ To develop and run applications that use DB2 .NET Data Provider, the .NET Framework Version 2.0 or 1.1 is required. The .NET Framework version 2.0 is recommended for new DB2 solutions development.
- ▶ For Visual Studio 2005 add-in, Microsoft Visual Studio 2005 is required and DB2 Application Development Client V8.2 fix pack 10 or later.
- ▶ For Visual Studio 2005 add-in, Windows XP, Service Pack 2, or later, or Windows Server® 2003, Service Pack 1.

DB2 .NET Data Provider can be used to connect to the following data sources:

- ▶ DB2 Universal Database for Linux, UNIX, and Windows, Version 9
- ▶ DB2 Universal Database Version 8 for Windows, UNIX, and Linux-based computers
- ▶ DB2 Universal Database Version 6 (or later) for OS/390 and z/OS, through DB2 Connect
- ▶ DB2 Universal Database Version 5, Release 1 (or later) for AS/400 and iSeries, through DB2 Connect

To ensure that the ADO.NET applications built for DB2 UDB V8 can function successfully with DB2 9, they need to be migrated. This may involve rebuilding the application if DB2 .NET Data Provider is used. An application built using OLE DB .NET Data Provider or ODBC .NET Data Provider does not need to be rebuilt.

For further details, refer to the following URL:

<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.uprun.doc/doc/t0023417.htm>

6.3 Add-in features for Visual Studio .NET

The IBM DB2 Development Add-In integrates a collection of DB2 specific features with the Microsoft Visual Studio .NET development environment.

Visual Studio .NET 2005 Add-In is available as a separate install from the core DB2 product. This was done to allow the tool to be available with other IBM databases.

The minimum requirement for the Visual Studio .NET 2005 Add-In is the DB2 Application Development Client V8.2 Fix Pack 10 or later.

The main features of IBM DB2 Development Add-In for V8 are:

- ▶ Launch various DB2 development and administration tools.
- ▶ Access and manage DB2 data connections in the IBM Explorer.
- ▶ Create and manage DB2 projects in the Solution Explorer.
- ▶ Create and modify DB2 scripts to create stored procedures and user defined functions (UDFs).

In Visual Studio 2005 Add-In, the following features are available in addition to the previous V8 IBM DB2 Development Add-In:

- ▶ Seamless integration with Microsoft Server Explorer, where DB2 connections can now be added into Microsoft Server Explorer (in V9, IBM Explorer has been deprecated).
- ▶ Introduction of IBM Designer to script and create DB2 database objects.
- ▶ IBM Database Add-Ins for Visual Studio 2005 is now available as a separate install.
- ▶ You can build Windows applications and Web sites for DB2 without writing any code:
 - All flavors of DB2 are supported (DB2 UDB for Linux, UNIX, and Windows, DB2 UDB for iSeries, and DB2 UDB for z/OS).
 - Federated database and nicknames are supported for application development.
 - Filtering of database objects is supported for optimal performance on iSeries and zSeries® servers.
 - Caching of schema information for objects in DB2 connections in Server Explorer provides better application development performance at design time.
- ▶ A new feature in IBM tooling is the ability to hide or show specific folders for a DB2 connection in Server Explorer.
- ▶ The tooling continues to support displaying detailed messages of DB2 activity on the IBM Message Pane.
- ▶ The new tooling introduces a new set of IBM designers to create, alter, and clone database objects:

- All IBM designers continue to use the smart multi-line editors that provide syntax colorization and statement completion.
- You have the ability to create new tables, views, and procedures using IBM designers.
- There is new functionality to alter existing tables, views, and procedures, using IBM designers.
- There is new functionality to create and alter roles and assign privileges to database objects.
- The IBM designers give you the ability to clone tables and procedures.
- ▶ The new tools give you a new way to seamlessly debug SQL procedures on Linux, UNIX, and Windows, or zSeries servers, from Server Explorer. Debugging support now uses the new IBM Designer for Procedures that allows a seamless debugging experience.
- ▶ There is a new designer to view or create script for all objects. The IBM Script Designer provides:
 - The ability to change and execute scripts.
 - The ability to run single or multiple DDL/DML statements and view results in single or multiple grids.
 - The ability to alter objects using scripts.
- ▶ You can show data from tables and views with the following new enhancements:
 - You can filter columns while retrieving data.
 - You can save data as XML to import or export, allowing easy table or view data migration.
- ▶ Execute procedures and functions has the following new enhancements:
 - You have the ability to run pre- and post-scripts.
 - You can save input or in-out parameter values across Visual Studio sessions.
 - You can commit or roll back transactions.
- ▶ There is a new user interface to view result sets in DB2 connections on the Server Explorer. It gives you:
 - The ability to view single or multiple result sets for a procedure in Server Explorer.
 - The ability to discover automatically (when possible) or to manually define or customize result set definitions for a procedure.

- The ability to set the preference to always discover or always manually define the result set definition in Add or Modify connection.
- ▶ You have continued support for DB2 projects and IBM Scripting wizards to create DB2 scripts. The debugging support has been discontinued from DB2 Projects. Instead, it is supported seamlessly from DB2 connections on Server Explorer.

6.3.1 Visual Studio 2005 Add-In: Server Explorer integration

DB2 connections can be added from Server Explorer using the Add Connection option.

1. Open Server Explorer if it is not already open using **View** → **Server Explorer**.
2. In the Server Explorer, right-click and select **Add Connection** from Data Connections node as shown in Figure 6-2.

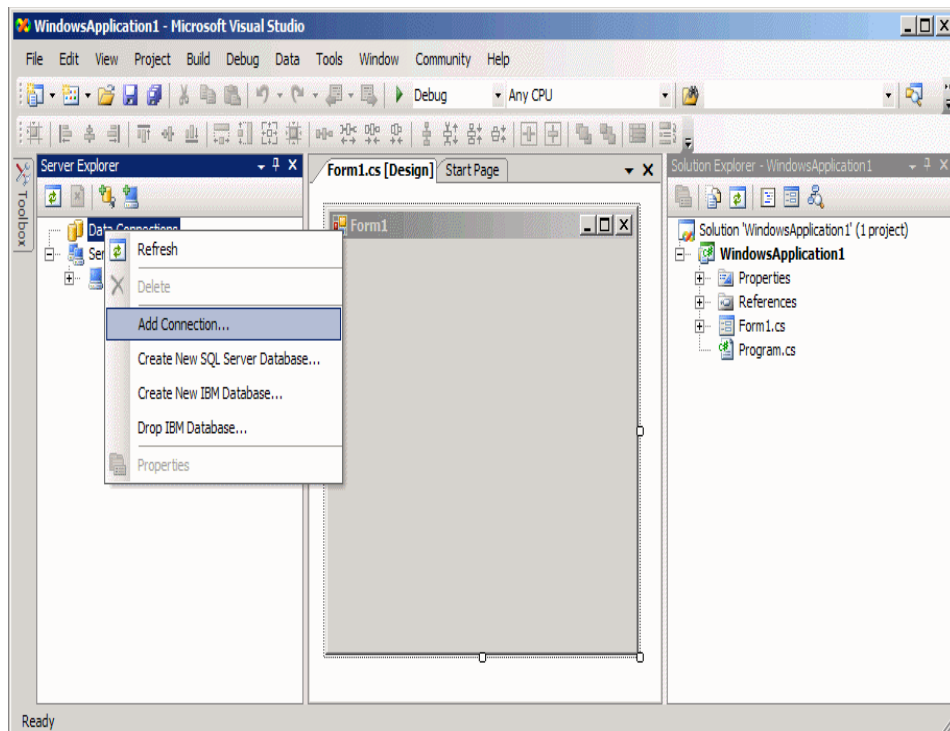


Figure 6-2 Adding connection in Data Explorer

3. Select **Data source** then select **IBM DB2** as shown in Figure 6-3. Click **Continue**.

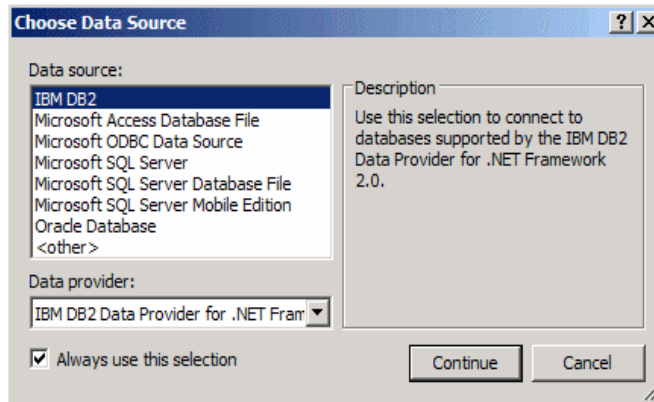


Figure 6-3 Selecting IBM DB2 in Data Explorer

4. Add the server name (given drop-down box will show server options), database name, user ID, and password as shown in Figure 6-4 on page 258. Click **OK**, which will now add the given database connection to Server

Explorer and
you can test the connection.

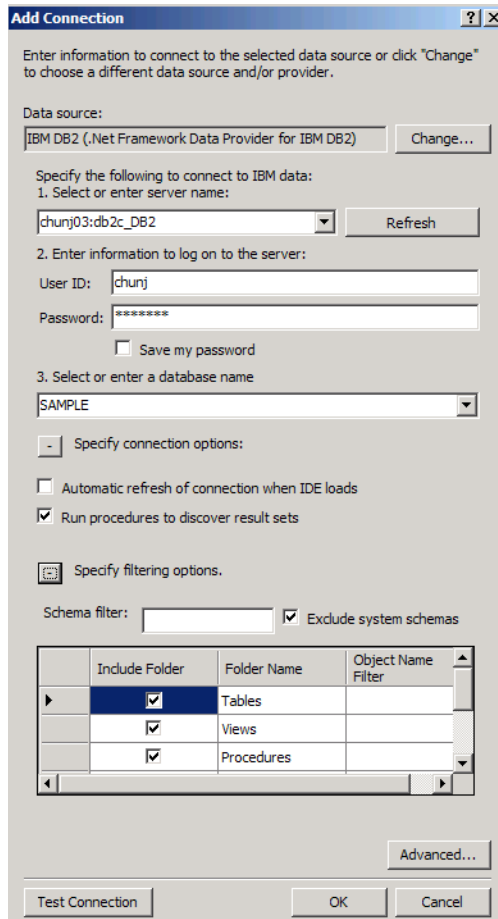


Figure 6-4 Providing connection information in Data Explorer

If the window shown in Figure 6-4 does not display and another window results in "Failed to find or load the registered .Net Framework Data Provider", then the following steps are needed to resolve the error:

- Open the DB2 Command window. Click **Start** → **Programs** → **IBM DB2** → **Command Line Tools** → **Command Window**.
- At the prompt in the DB2 Command window, enter the following command:

```
db2lswtch.exe -promote
```

Repeat the procedure to add DB2 connection.

6.3.2 Visual Studio 2005 Add-In: IBM Designer

IBM Designer provides tools to create various database objects. It can be used to create and alter tables, views, scripts, and procedures.

Note: The wizard and designer can both be used to create the same database objects. The wizard, however, is targeted to novice users who prefer step-by-step creation while the designer allows a more advanced user to switch between views and other applications in the design process.

IBM Table Designer

The IBM Table Designer can be used to create a new table. From Server Explorer, chose **Data Connections** → **database**, right-click **Tables** node, and select **Add New Table with Designer** as shown in Figure 6-5.

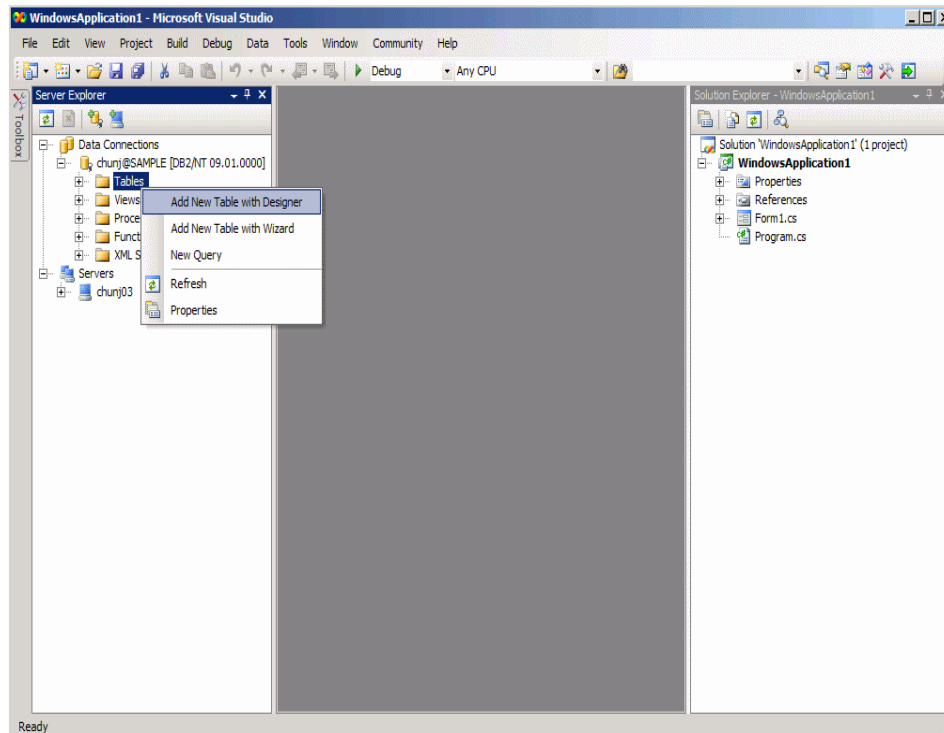


Figure 6-5 Starting IBM Table Designer

The Table Designer window shows Table Definition, Columns, and Column properties section as in Figure 6-6.

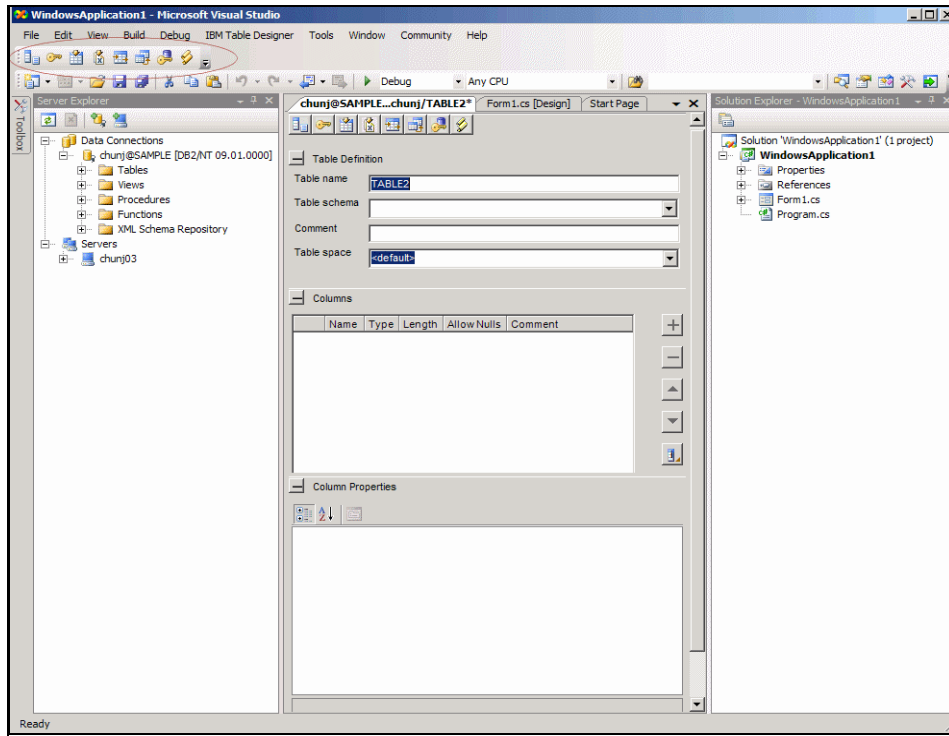


Figure 6-6 Table Designer window

The following menus are provided in Table Designer window:

- ▶ Column Definitions View (Default): Allows you to define columns and their data types.
- ▶ Keys View: Allows defining primary keys, unique keys, and foreign keys.
- ▶ Indexes View: Allows creating indexes for the columns defined in the columns view.
- ▶ XML Indexes View: Allows creating indexes for any tags in XML document.
- ▶ Check Constraints View: Allows adding check constraints for the table and its columns.
- ▶ Triggers View: Allows adding triggers to a table.
- ▶ Privileges View: Allows adding roles and privileges to the users.
- ▶ Show Script View: This shows the create table statement command.

IBM View Designer

The IBM View Designer can be used to create a new view by right-clicking **Views** node in Server Explorer under **Data Connections** → **database**, and selecting **Add New View with Designer** as shown in Figure 6-7.

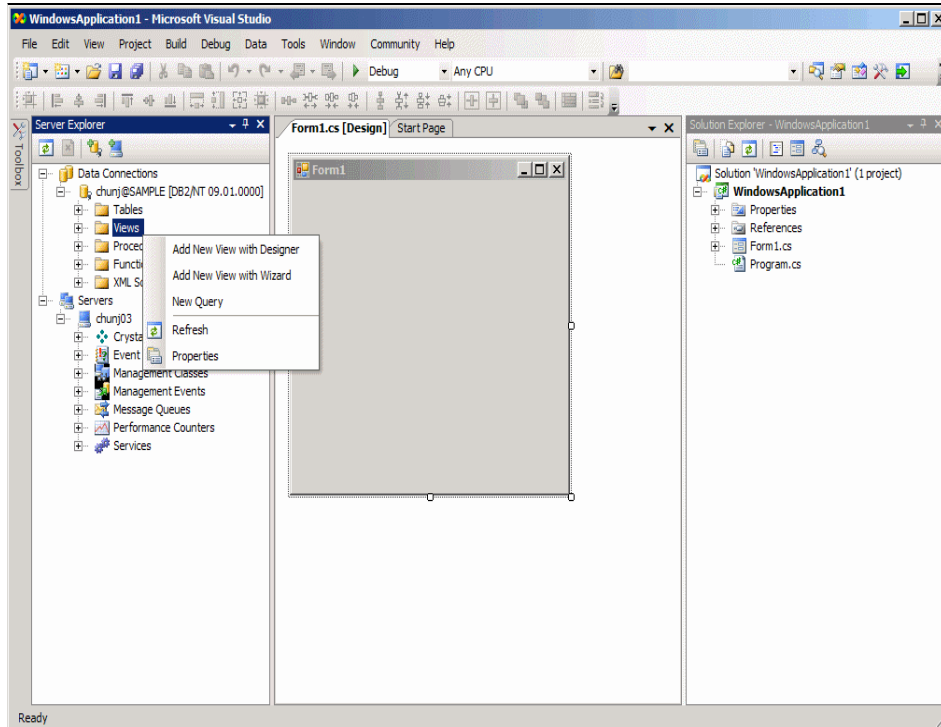


Figure 6-7 Starting IBM View Designer

The View Designer window is shown in Figure 6-8.

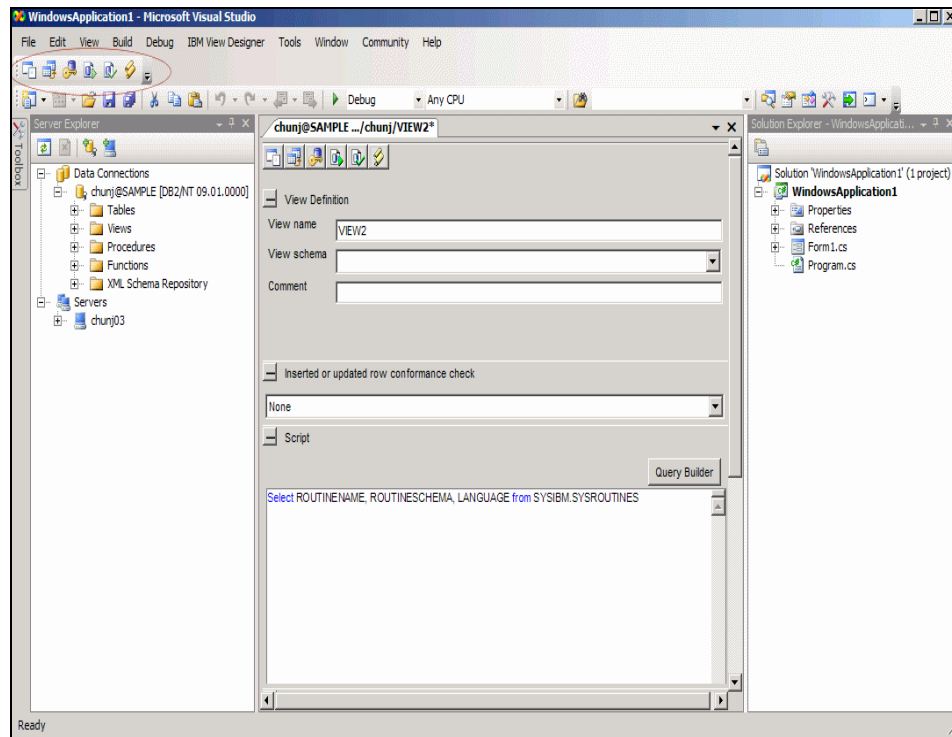


Figure 6-8 View Designer window

The menu provided in View Designer window includes the following:

- ▶ View View: Provides ability to check the syntax or test execute the SQL statement provided in the view definition.
- ▶ Triggers View: Provides ability to define and add triggers.
- ▶ Privileges View: Allows adding roles and privileges to users.
- ▶ Show Script View: Shows the create view statement.

IBM Procedure Designer

The IBM Procedure Designer can be used to create a new procedure by right-clicking **Procedures** node in Server Explorer, under **Data Connections** → **database**, then selecting **Add New SQL Procedure with Designer** as shown in Figure 6-9 on page 263.

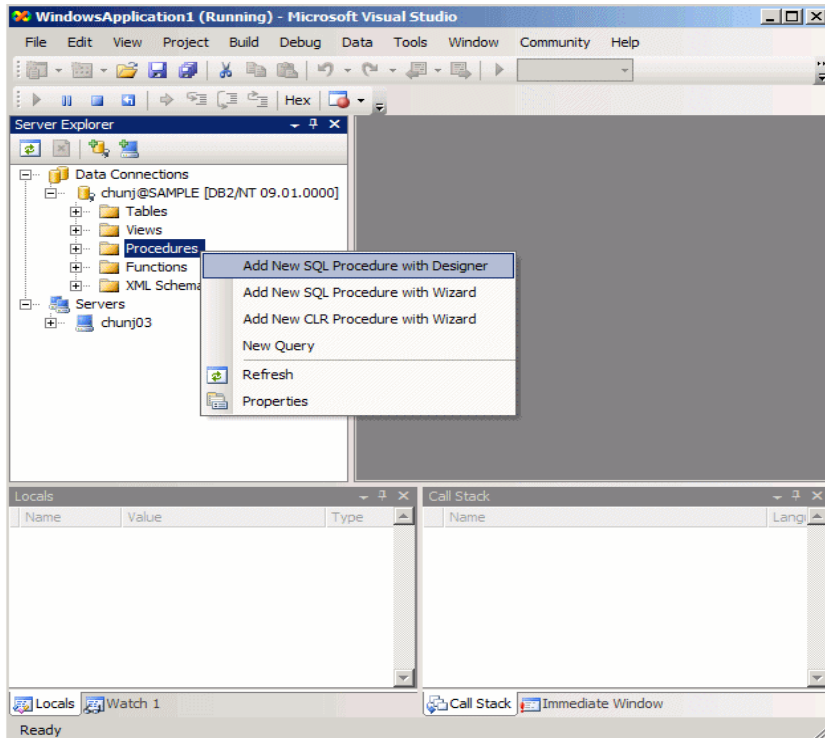


Figure 6-9 Starting IBM Procedure Designer

The Procedure Designer window is shown in Figure 6-10.

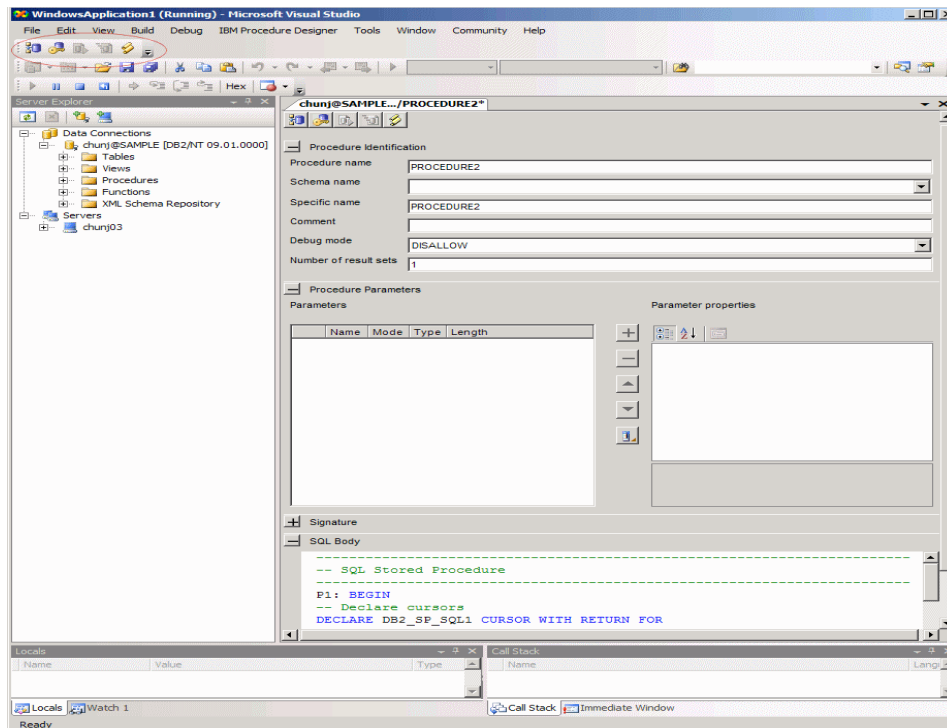


Figure 6-10 Procedure Designer window

Note: At time of DB2 9 release, CLR Procedure can only be created using the wizard. Only SQL Procedure Designer is available.

The menu provided in Procedure Designer window includes the following:

- ▶ Procedure View: Allows defining procedure parameters and procedure body.
- ▶ Privileges View: Allows adding roles and privileges to the users or groups.
- ▶ Show Script View: Shows the create procedure statement.

Debugging SQL procedures

Debugging support for SQL procedure has been extended to zSeries servers.

Procedure can be debugged using the following steps:

1. Create SQL procedure with Debug mode option set to ALLOW.
2. Add breakpoints in SQL body.

This can be done using **Debug** → **Toggle Breakpoint** (F9), which will highlight the line and place a red dot to the left of the breakpoint as shown in Figure 6-11 on page 266.

Breakpoints can be deleted by left-clicking the **red dot**.

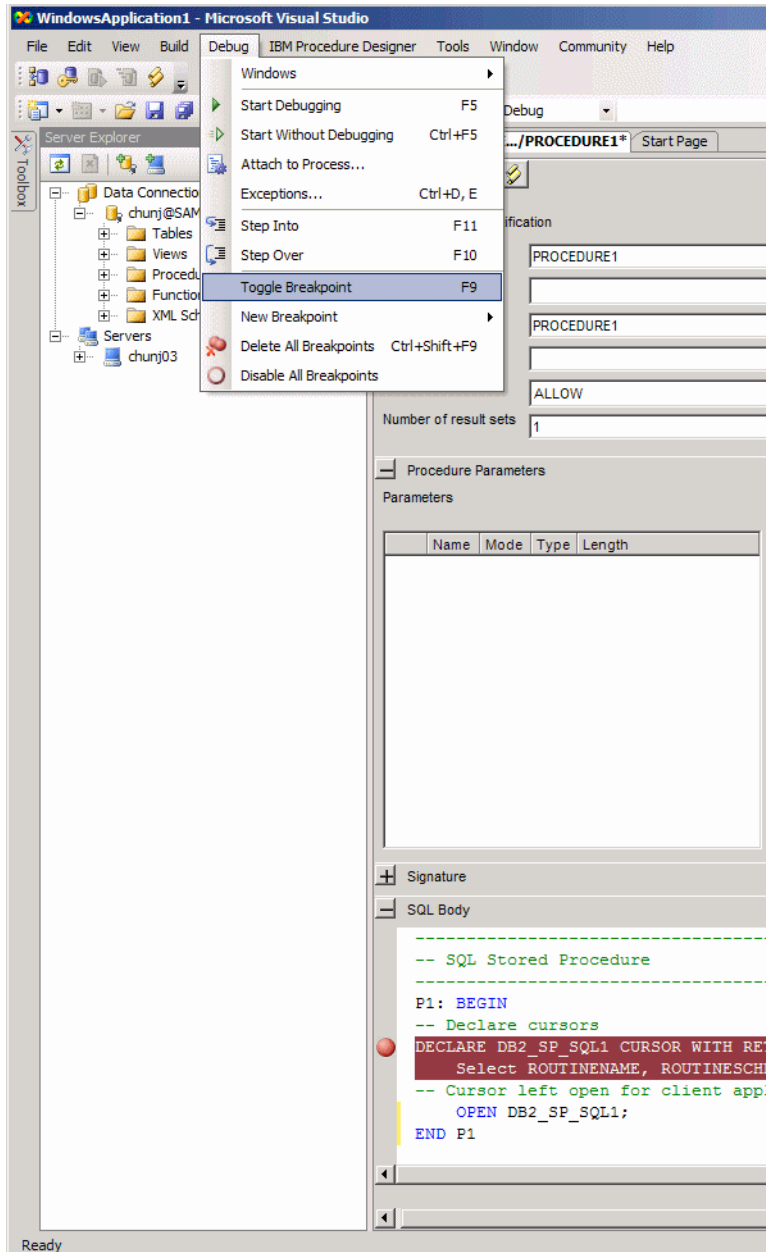


Figure 6-11 Adding debug breakpoints in Procedure Designer

3. Start debugging using **Step Into** button on the toolbar.

6.4 Data Providers for ADO.NET

In ADO.NET architecture, applications (also called Data Consumers) connect to database (also referred to as Resource) using data provider. The data provider encapsulates data and provides a means to interact with the database including connection, execution of SQL command, and retrieval of results.

As mentioned in Chapter 1, “DB2 application development overview” on page 1, IBM DB2 provides three data providers for ADO.NET applications. These are:

- ▶ DB2 .NET Data Provider
- ▶ OLE DB .NET Data Provider
- ▶ ODBC .NET Data Provider

6.4.1 Managed provider and unmanaged provider

In ADO.NET, Data Provider can be separated into two categories according to how it is implemented:

- ▶ Managed Provider: This executes within the ADO.NET environment, which controls all aspects of application execution, including memory allocation, memory deallocation, application domains, and so on.
- ▶ Unmanaged Provider: This is pre-ADO.NET Windows 32-bit operating system-based drivers.

DB2 .NET Data Provider is managed type. DB2 .NET Data provider is ADO.NET data provider, which is recommended for use with DB2 family databases. The following namespaces are required for DB2 .NET Data Provider:

```
using System;  
using System.Data;  
using IBM.Data.DB2;
```

OLE DB .NET Data Provider is a bridge provider that passes the ADO.NET request to native IBM OLE DB provider (IBMDADB2). The following namespaces are required for OLE DB .NET Data Provider:

```
using System;  
using System.Data;  
using System.Data.OleDb;
```

ODBC .NET Data Provider is a bridge provider that passes ADO.NET requests to the IBM ODBC Driver. The following namespaces are required for ODBC .NET Data Provider:

```
using System;  
using System.Data;
```

```
using System.Data.Odbc;
```

Use of OLE DB .NET Data Provider or ODBC .NET Data Provider is recommended if the application is connecting to multiple vendor databases and you do not wish to change any code within the application.

DB2 .NET Data Provider is recommended for any new ADO.NET application development. It will yield the best performance due to the elimination of the extra unmanaged layer as shown in Figure 6-12.

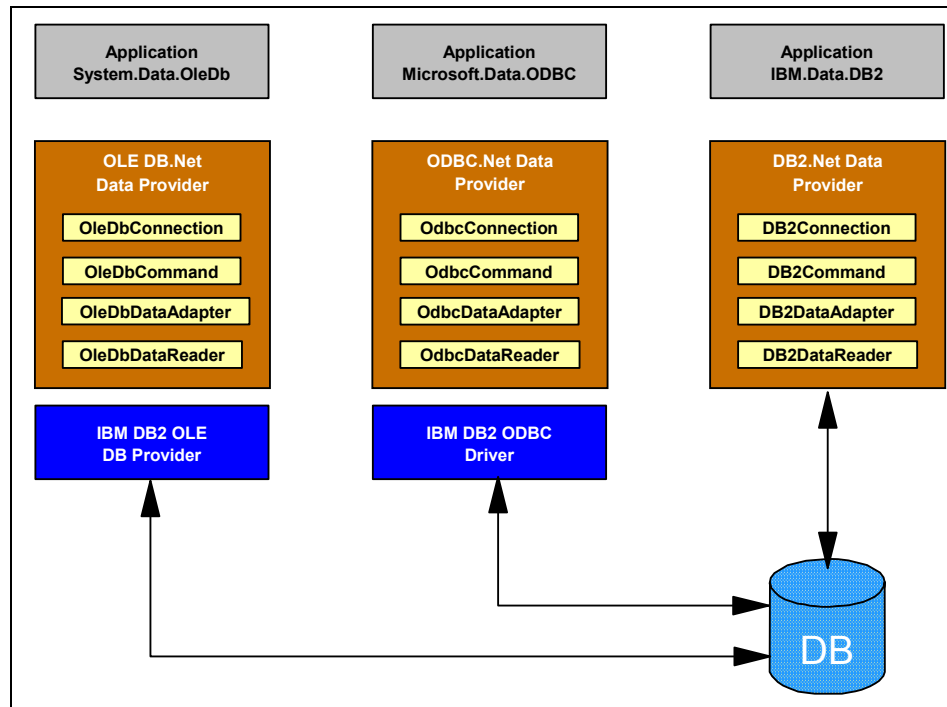


Figure 6-12 DB2 Data Provider

Five objects in ADO.NET make up core functionalities. This is illustrated in Figure 6-1 on page 252.

Five key managed provider components in ADO.NET

There are five key managed provider components in ADO.NET that are common to all IBM DB2 data providers. These include:

- ▶ Connection
- ▶ Command
- ▶ DataReader
- ▶ DataAdapter

- ▶ DataSet

Connection

The *Connection* object is used to connect to a database and control the transactions in ADO.NET. Each data provider has different Connection Objects. Each of the three data providers IBM DB2 incorporates has its own Connection Objects (DB2Connection, OleDbConnection, and OdbcConnection).

The data connection class enables you to specify the connection string used to connect to the target database server:

- ▶ This is implemented as DB2Connection object in DB2 .NET Data Provider, for example:

```
DB2Connection connection = new DB2Connection("Database=SAMPLE");
```

- ▶ This is implemented as OleDbConnection object in DB2 OLE DB Data Provider, for example:

```
OleDbConnection connection = new  
OleDbConnection("Provider=IBMDADB2.1;DSN=SAMPLE");
```

- ▶ This is implemented as OdbcConnection object in DB2 ODBC Data Provider, for example:

```
OdbcConnection connection = new OdbcConnection("DSN=SAMPLE");
```

The connection object has public property *ConnectionString*, which is required for establishing connection with a database. It requires database name and other parameters such as user ID and password, for example:

```
connection.ConnectionString = "Database=Sample";
```

However, ConnectionString property can be set through passing the string to the Connection Object Constructor as shown in following example:

```
DB2Connection connection = new DB2Connection("Database=SAMPLE");
```

Connection objects have the following public methods:

- ▶ **Open:** This opens a database connection as specified in a ConnectionString, for example:

```
connection.Open();
```

Connections can be opened by explicitly calling the Open method on the connection as shown above or by implicitly using a DataAdapter.

- ▶ **Close:** This closes the database connection, for example:

```
connection.Close();
```

- ▶ **CreateCommand:** This returns a command object associated with the connection, for example:

```
connection.CreateCommand();
```

- ▶ **BeginTransaction:** This begins the database transaction, for example:

```
connection.BeginTransaction();
```

Command

The Command object allows for execution of any supported SQL statement or stored procedure using a data connection object. Connection object should be created but do not needed to be opened prior to creating SQL commands:

- ▶ This is implemented as DB2Command in DB2 .NET Data Provider, for example:

```
DB2Command cmd = new DB2Command();
```

- ▶ This is implemented as OleDbCommand in DB2 OLE DB Data Provider, for example:

```
OleDbCommand cmd = new OleDbCommand();
```

- ▶ This is implemented as OdbcCommand in DB2 ODBC Data Provider, for example:

```
OdbcCommand cmd = new OdbcCommand();
```

The Command object has public properties *CommandType* and *CommandText*. The CommandType describes whether an SQL statement or a stored procedure will be executed. The CommandText is used to set or get an SQL statement or a stored procedure that is to be executed, for example:

```
cmd.CommandType = CommandType.Text;  
cmd.CommandText = "SELECT manager FROM org WHERE DEPTNUMB=10";
```

or

```
cmd.CommandType = CommandType.StoredProcedure;  
cmd.CommandText = procName;
```

Command object has the following public methods:

- ▶ **CreateParameter:** This is used for parameter handling, for example:

```
set param1 = cmd.CreateParameter("DEPTNAME", adVarChar,  
adParamInput, 14, "Test");  
set param2 = cmd.CreateParameter("DEPTNUMB", adTinyInt,  
adParamInput, , 510);
```

- ▶ **ExecuteNonQuery:** Use this to execute a SQL command that does not return any data, such as UPDATE, INSERT, or DELETE SQL operations. Method returns the number of rows affected for given execution, as shown:


```
int rowsAffected = cmd.ExecuteNonQuery();
```
- ▶ **ExecuteReader:** Use this to execute a SQL query that returns a DataReader. DataReader is fast forward-only stream of data, for example:


```
DB2DataReader reader = cmd.ExecuteReader ( );
```
- ▶ **ExecuteScalar:** Use this to execute a SQL command that retrieve a single value from a database, for example:


```
int count=(int)cmd.ExecuteScalar();
```

Note: Object returned by cmd.ExecuteScalar() should be casted to data type of underlying database object. The above example is valid for a case where a single value being retrieved is from int column.

DataAdapter

The data adapter object populates a disconnected DataSet with data and performs update. It contains the four optional commands for the select, insert, update, and delete. Use it between DataSet and database for loading and unloading data.

- ▶ Implement this as DB2DataAdapter in DB2 .NET Data Provider, for example:


```
DB2DataAdapter adapter = new DB2DataAdapter();
```
- ▶ Implement this as OleDbDataAdapter in DB2 OLE DB Data Provider, for example:


```
OleDbDataAdapter adapter = new OleDbDataAdapter();
```
- ▶ Implement this as OdbcDataAdapter in DB2 ODBC Data Provider, for example:


```
OdbcDataAdapter adapter = new OdbcDataAdapter();
```

Data adapter object has these public properties:

- ▶ DeleteCommand
- ▶ InsertCommand
- ▶ SelectCommand
- ▶ UpdateCommand

The *DeleteCommand* deletes records using SQL statements or stored procedures from the data set, for example:

```
adapter.DeleteCommand = new DB2Command("DELETE From org WHERE DEPTNUMB = 10", connection);
```

The *InsertCommand* inserts new records into a database using SQL or stored procedures, for example:

```
adapter.InsertCommand = new DB2Command("INSERT INTO org VALUES (30, 'Test', 60, 'Eastern', 'Toronto')", connection);
```

The *SelectCommand* selects records in a database using SQL or Stored Procedures, for example:

```
adapter.SelectCommand = new DB2Command("SELECT manager From org WHERE DEPTNUMB = 30", connection);
```

The *UpdateCommand* update records in a database using SQL or Stored Procedures, for example:

```
adapter.UpdateCommand = new DB2Command("UPDATE org SET Manager=70 WHERE DEPTNUMB=20", connection);
```

Data Adapter has the following public methods:

- ▶ **Fill:** This fills records in DataSet, as shown below:

```
DataSet results= new DataSet();
adapter.SelectCommand = new DB2Command("Select * from dept",
connection);
adapter.Fill(results);
```

- ▶ **Update:** This updates Records in DataSet and a database through INSERT, UPDATE, and DELETE operations, for example:

```
DataSet results= new DataSet();
adapter.UpdateCommand = new DB2Command("UPDATE org SET Manager=70
WHERE DEPTNUMB=20", connection);
adapter.Update(results);
```

DataReader

Utilized for fast forward-only, read-only access to connected record sets that are returned from executing SQL statements or stored procedure calls. The DataReader object cannot be directly instantiated and needs to be returned as the result of the Command Object's ExecuteReader method.

- ▶ Implement this as DB2DataReader in DB2 .NET Data Provider, for example:

```
Db2DataReader reader = cmd.ExecuteReader();
```

- ▶ Implement this as OleDbDataReader in DB2 OLE DB Data Provider, for example:

```
OleDbDataReader reader = cmd.ExecuteReader();
```


- ▶ Implement this as `OdbcDataReader` in DB2 ODBC Data Provider, for example:

```
OdbcDataReader reader = cmd.ExecuteReader();
```

The `DataReader` object has *FieldCount* and *HasRows* public properties. The `FieldCount` property returns the total number of columns in the current row while `HasRows` property indicates whether `DataReader` has one or more rows by returning true or false, for example:

```
int cols=reader.FieldCount;
bool rows=reader.HasRows;
```

The `DataReader` object has the following public methods:

- ▶ **Read:** Reads in records one row at a time and advances the cursor to the next row. It returns true or false to indicate whether there are any rows to read, for example:

```
bool done=reader.read();
```

- ▶ **Close:** This closes the `DataReader`, for example:

```
reader.Close();
```

- ▶ **Getxxx:** This is used to get data of type xxx, for example:

```
Console.WriteLine (reader.GetString(1));
```

DataSet

The `DataSet` object represents an “In-memory cache of data”, which was retrieved from the database. The `DataSet` object is a disconnected dataset, which provides a consistent relational programming model independent of the data source. Since it is disconnected from the database, it reduces the communication overhead to the database server.

The `DataSet` object has the public property *DataSetName*, which gets or sets `DataSet` name, for example:

```
DataSet ds = new DataSet();
ds.DataSetName = "DB2";
```

The `DataSet` object has the following public methods:

- ▶ **AcceptChanges:** This commits changes to the `DataSet`, for example:

```
ds.AcceptChanges();
```

- ▶ **Clear:** This clears the `DataSet` contents, for example:

```
ds.Clear();
```

- ▶ **GetXML**: This gets XML representation of data in the DataSet, for example:
`Console.WriteLine(ds.GetXml())`
- ▶ **ReadXML**: This reads XML schema and XML into DataSet, for example:
`ds.ReadXML(reader);`
- ▶ **WriteXML**: This writes XML schema and XML into DataSet, for example:
`ds.WriteXml (".\\test.xml") ;`

The short ADO.NET sample C# codes shown in Example 6-1 demonstrate the use of various DB2 Data Providers.

They perform the same functionality where “selects * from staff” query is issued and displays the name of the staff (second column in staff table) to the screen. These sample codes require DB2 Sample database. Key differences between the codes are highlighted in bold.

DB2 .NET Data Provider C# sample is shown in Example 6-1.

Example 6-1 Short C# sample code using DB2 .NET Data Provider

```
using System;
using System.Data;
using IBM.Data.DB2;

class NETSamp
{
    public static void Main(String[] args)
    {
        DB2Connection conn = null;
        DB2Command cmd = null;
        DB2DataReader reader = null;
        int cols=0;
        bool rows=false;

        try{
            conn=new DB2Connection("Database=SAMPLE");
            Console.WriteLine("\n Connecting to the database.");
            // Opening the connection
            conn.Open();
            // Create the command to be executed
            cmd=conn.CreateCommand();

            //Prepare the query CommandText.
            cmd.CommandText = "SELECT * FROM staff";
        }
    }
}
```

```

// Retrieve and display the single value
reader = cmd.ExecuteReader();
Console.WriteLine("\nExecute: "+cmd.CommandText);
// Checking to for number of columns and see if any rows
//are returned.
cols= reader.FieldCount;
Console.WriteLine("\n FieldCount: "+cols);
rows=reader.HasRows;
Console.WriteLine("\n HasRows?: "+rows);

while (reader.Read()==true)
{
    //Read the second column which contains staff name
    Console.WriteLine (reader.GetString(1));
}
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}finally{
    //clean up the resources by closing reader and connection
    reader.Close();
    conn.Close();
}
}
}

```

DB2 OLE DB Data Provider C# sample is shown in Example 6-2.

Example 6-2 Short C# sample code using DB2 OLE DB Data Provider

```

using System;
using System.Data;
using System.Data.OleDb;

class OLEDBSamp
{
    public static void Main(String[] args)
    {
        OleDbConnection conn = null;
        OleDbCommand cmd = null;
        OleDbDataReader reader = null;
        int cols=0;
        bool rows=false;
        try{

```

```

conn = new OleDbConnection("Provider=IBMDADB2.1;DSN=SAMPLE");
Console.WriteLine("\n Connecting to the database.");
// Opening the connection
conn.Open();
// Create the command to be executed
cmd=conn.CreateCommand();
//Prepare the query CommandText.
cmd.CommandText = "SELECT * FROM staff";
// Retrieve and display the single value
reader = cmd.ExecuteReader();
Console.WriteLine("\nExecute: "+cmd.CommandText);
// Checking to for number of columns and see if any rows
//are returned.
cols= reader.FieldCount;
Console.WriteLine("\n FieldCount: "+cols);
rows=reader.HasRows;
Console.WriteLine("\n HasRows?: "+rows);
while (reader.Read()==true)
{
    //Read the second column which contains staff name
    Console.WriteLine (reader.GetString(1));
}
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}finally{
    //clean up the resources by closing reader and connection
    reader.Close();
    conn.Close();
}
}
}

```

DB2 ODBC Data Provider C# sample is shown in Example 6-3.

Example 6-3 Short C# sample code using DB2 ODBC Data Provider

```

//Before starting make sure SAMPLE database entry is present in System
DSN tab of Control Panel->Data Sources (ODBC).
using System;
using System.Data;
using System.Data.Odbc;

class ODBCSamp

```

```

{
public static void Main(String[] args)
{
    OdbcConnection conn = null;
    OdbcCommand cmd = null;
    OdbcDataReader reader = null;
    int cols=0;
    bool rows=false;

    try{
        conn = new OdbcConnection("DSN=SAMPLE");
        Console.WriteLine("\n Connecting to the database.");
        // Opening the connection
        conn.Open();
        // Create the command to be executed
        cmd=conn.CreateCommand();

        //Prepare the query CommandText.
        cmd.CommandText = "SELECT * FROM staff";

        // Retrieve and display the single value
        reader = cmd.ExecuteReader();
        Console.WriteLine("\nExecute: "+cmd.CommandText);
        // Checking to for number of columns and see if any rows
        //are returned.
        cols= reader.FieldCount;
        Console.WriteLine("\n FieldCount: "+cols);
        rows=reader.HasRows;
        Console.WriteLine("\n HasRows?: "+rows);
        while (reader.Read()==true)
        {
            //Read the second column which contains staff name
            Console.WriteLine (reader.GetString(1));
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }finally{
        //clean up the resources by closing reader and connection
        reader.Close();
        conn.Close();
    }
}
}

```

```
}
```

The above samples can be compiled using the following command:

```
csc NETSamp.cs /r:<DB2 Install Path>\bin\netf20\IBM.Data.DB2.dll
```

The <DB2 Install Path> is the path where db2 is installed. Example 6-4 shows sample application output.

Example 6-4 Sample application output

Connecting to the database.

```
Execute: SELECT * FROM staff
```

```
FieldCount: 7
```

```
HasRows?: True
```

```
Sanders
```

```
Pernal
```

```
Marenghi
```

```
O'Brien
```

```
Hanes
```

```
...
```

Closing reader and disconnecting from the database.

In Microsoft .NET Framework, classes are organized into a hierarchical structure of related groups called *namespaces*. System.Data namespace contains classes associated with the use of ADO.NET.

In C#, all errors are treated as instances of an exception.

Error handling in ADO.NET is done in the form of the *try/catch/finally* or the *On Error* construct.

Summary

Table 6-1 shows the summary various DB2 data providers comparison.

Table 6-1 DB2 Data Provider comparison

	DB2 .NET Data Provider	DB2 OLE DB Data Provider	DB2 ODBC Data Provider
Namespace	IBM.Data.DB2	System.Data.OleDb	System.Data.ODBC

	DB2 .NET Data Provider	DB2 OLE DB Data Provider	DB2 ODBC Data Provider
Connection Object (Used to create connection)	DB2Connection	System.Data.ODBC	ODBCConnection
Command Object (Used to execute command)	DB2Command	OleDbCommand	ODBCCommand
DataReader Object (Used to read retrieved data)	DB2DataReader	OleDbDataReader	ODBCDataReader
DataAdapter Object (Used to load and unload data between DataSet and database)	DB2DataAdapter	OleDbDataAdapter	ODBCDataAdapter
ConnectionString	Database= SAMPLE	PROVIDER=IBMDA DB2;DSN=SAMPLE ;UID=db2admin;PW D=db2admin	DSN=SAMPLE;UID =db2admin; PWD=db2admin; DRIVER={IBM DB2 ODBC Driver}
Transaction Object	DB2Transaction	OleDbTransaction	ODBCTransaction

OLE DB Data provider and ODBC Data Provider have limitations. Refer to following URLs for complete listings:

- ▶ OLE DB Data provider limitations

<http://publib.boulder.ibm.com/infocenter/db21uw/v9/index.jsp?topic=/com.ibm.db2.udb.apdv.ms.doc/doc/r0011826.htm>

- ▶ ODBC Data Provider limitations

<http://publib.boulder.ibm.com/infocenter/db21uw/v9/index.jsp?topic=/com.ibm.db2.udb.apdv.ms.doc/doc/r0011829.htm>

6.5 Application example using ADO.NET

In this section, we provide sample C# application codes, which utilize IBM DB2 .NET Data provider to demonstrate inserting new customer data into the

customer table as well as updating and deleting existing customer information in the SAMPLE database.

Note: The CUSTOMER table is only present in DB2 9 and it has two XML data type columns.

The CUSTOMER table in the SAMPLE database has three columns as defined in Table 6-2. The primary key consists of CID (customer ID) column.

Table 6-2 CUSTOMER table from sample database

Column name	Type name	Length	Scale	Nullable
CID	BIGINT	8	0	No
INFO	XML	0	0	Yes
History	XML	0	0	Yes

In the following set of samples, each task has been presented as a separate method.

Establishing the connection to the database

Establishing connection to the database is the essential first step. The connection code to IBM DB2 database using DB2 .NET data provider is implemented in `ConnectDb()` method shown in Example 6-6 on page 281.

The `ConnectDb()` method will return `DB2Connection` Object once connection has been established using the three parameters (db alias, userid, and password) required for the connection string. If passed database alias is null, it will establish default connection to the SAMPLE database.

Connection is established by first creating `DB2Connection` object and passing the connection string to the constructor:

```
DB2Connection conn = new DB2Connection(connectionString);
```

The namespaces shown in Example 6-5 are required for given samples:

Example 6-5 Namespaces for IBM DB2 .NET Data Provider

```
using System;  
using System.Data;  
using IBM.Data.DB2; //needed for IBM DB2 .NET Data Provider  
using System.Xml;
```


Example 6-6 Connecting to the DB2 database using .NET data provider

```
public static DB2Connection ConnectDb(String alias, String userid,
String password )
{
    String dsn = "SAMPLE";
    String connectString;

    //Since this method will be called internally,
    //we don't have to worry about all possible connect string
    scenario.
    if(alias == null)
    {

        connectString = "Database="+dsn;
    }
    else
    {
        dsn = alias;
        connectString = "Database=" + dsn + ";UID=" + userid + ";PWD=" +
            password;
    }
    DB2Connection conn = new DB2Connection(connectString);
    try{
        conn.Open();
        Console.WriteLine(" Connected to the " + dsn + " database");
    }catch (Exception e){
        Console.WriteLine(e.Message);
    }
    return conn;
} // ConnectDb
```

Selecting existing customer information

In order to read the customer data from CUSTOMER table, we select INFO column, which contains customer information using the WHERE Clause with specific CID.

The sample INFO XML data from CUSTOMER table is shown in Example 6-7.

Example 6-7 Info XML data from the CUSTOMER table

```
<customerinfo xmlns="http://posample.org" Cid="1000">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
```

```
        <prov-state>Ontario</prov-state>
        <pcode-zip>M6W 1E6</pcode-zip>
    </addr>
    <phone type="work">416-555-1358</phone>
</customerinfo>
```

The customer ID (CID) is also stored as an attribute in customerinfo element.

The INFO column being returned from the SELECT statement is XMLdata type, which will be read using XmlReader object.

The XmlReader object is an event-based, read-only, forward-only XML pull parser. It provides functionality for reading in XML documents.

```
XmlReader xmlreader = cmd.ExecuteXmlReader();
```

The XmlReader object is then loaded into XmlDocument object, which represents XML document as a node tree, where elements and attributes are stored as nodes that contain relational information.

```
XmlDocument.Load(xmlreader);
```

Presence of XML namespace determines whether we need to add XmlNamespaceManager in order to successfully select nodes.

```
XmlNamespaceManager nsmgr = new XmlNamespaceManager(XmlNameTable);
nsmgr.AddNamespace(prefix, uri);
```

If the XPath expression in SelectNodes does not include a prefix, it is assumed that the namespace URI is the empty namespace. If XML document includes a default namespace, it must be added to a prefix and namespace URI to the XmlNamespaceManager or none of the nodes will get selected. The XmlNamespaceManager is required to resolve any prefixes in the XPath expression.

```
selXmlDoc.SelectNodes("//prefix:element", nsmgr);
```

If XML namespace is not present in XML document (in absence of xmlns attribute), XmlNamespaceManager is not needed and XPath expression in SelectNodes will appear as following:

```
selXmlDoc.SelectNodes("//element");
```

The double slashes (//) in XPath above refer to descendant-or-self Axis, meaning it contains the context node in addition to all the nodes contained in the descendant axis.

Note that the use of (//) prefix will yield all instances of element name specified in the XML document.

If you want to differentiate elements of same name in different nodes, you can specify the specific XPath.

For example, in the CUSTOMER table, we have two instances of `<name>` in the XML document for the “info” column. One is for name of the customer and the other is for the assistant’s name as shown in Figure 6-13.

```
- <customerinfo xmlns="http://posample.org" Cid="1005">
  <name>Larry Menard</name>
  - <addr country="Canada">
    <street>223 NatureValley Road</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M4C 5K8</pcode-zip>
  </addr>
  <phone type="work">905-555-9146</phone>
  <phone type="home">416-555-6121</phone>
  - <assistant>
    <name>Goose Defender</name>
    <phone type="home">416-555-1943</phone>
  </assistant>
</customerinfo>
```

Figure 6-13 Sample info XML data from the CUSTOMER table

The XPath shown in Example 6-8 can be specified in the `SelectNodes()` to obtain all the names in the given XML document (customer’s and assistant’s).

Example 6-8 Selecting all names

```
XmlNodeList customername= selXmlDoc.SelectNodes("//cust:name", nsmgr);
foreach (XmlElement nameinfo in customername)
{
  String vname=nameinfo.InnerXml;
}
```

The XPath in Example 6-9 can be specified in the `SelectNodes()` to obtain customer’s name and assistant’s name respectively.

Example 6-9 Obtain customer’s name and assistant’s name respectively

```
//Customer’s name
XmlNodeList customername =
selXmlDoc.SelectNodes("/cust:customerinfo/cust:name", nsmgr);
foreach(XmlElement nameinfo in customername)
{
  String custname = nameinfo.InnerXml;
}
```

```

}
...
//Assistant's name
XmlNodeList assistantname =
selXmlDoc.SelectNodes("/cust:customerinfo/cust:assistant/cust:name",
nsmgr);
foreach (XmlElement anameinfo in assistantname)
{
    String assistname = anameinfo.InnerXml;
}

```

The value of the element is extracted using:

```
XmlElement.InnerXml(localname, XmlNamespaceManager).
```

The value of the attribute is extracted using:

```
XmlElement.GetAttribute(localname, XmlNamespaceManager).
```

The XmlNodeList implements IEnumerable interface, and thus it can be accessed using IEnumerator methods. The code, which selects INFO XML column from the customer table is implemented in SelectCustomer() method as shown in Example 6-10.

Example 6-10 Selecting customer information

```

public static void SelectCustomer(DB2Connection conn, int pcid)
{
    //Select the data according to cid.
    DB2Command cmd = conn.CreateCommand();
    XmlDocument selXmlDoc = new XmlDocument();
    String selectStmt="SELECT info from customer WHERE cid = " + pcid +
";" ;
    Console.WriteLine("\nSelect Stmt: \n"+selectStmt);
    cmd.CommandText= selectStmt;

    XmlReader xr = cmd.ExecuteXmlReader();
    try{
        //if following XmlNamespaceMangaer is not added no node will be
selected as given XML data contains xmlns
        XmlNamespaceManager nsmgr = new XmlNamespaceManager(xr.NameTable);
        nsmgr.AddNamespace("cust", "http://posample.org");

        Console.WriteLine("\nSelecting using xmlreader\n");

        selXmlDoc.Load(xr);
    }
}

```

```

        XmlNodeList customerinfo=
selXmlDoc.SelectNodes("//cust:customerinfo", nsmgr);
        foreach (XmlElement info in customerinfo)
        {
            String vcid=info.GetAttribute("Cid");
            Console.WriteLine("\n pcid: "+ pcid +"\n vcid: "+vcid);
        }

        XmlNodeList customername= selXmlDoc.SelectNodes("//cust:name",
nsmgr);
        foreach (XmlElement nameinfo in customername)
        {
            String vname=nameinfo.InnerXml;
            Console.WriteLine("\n pcid: "+ pcid +"\n vname: "+vname);
        }

        XmlNodeList customeraddr= selXmlDoc.SelectNodes("//cust:addr",
nsmgr);
        foreach (XmlElement addrinfo in customeraddr)
        {
            String country = addrinfo.GetAttribute("country");
            Console.WriteLine("\n pcid: "+ pcid +"\n country: "+country);
        }

        XmlNodeList customerstreet= selXmlDoc.SelectNodes("//cust:street",
nsmgr);
        foreach (XmlElement stinfo in customerstreet)
        {
            String street = stinfo.InnerXml;
            Console.WriteLine("\n pcid: "+ pcid +"\n street: "+street);
        }

        XmlNodeList customercity= selXmlDoc.SelectNodes("//cust:city",
nsmgr);
        foreach (XmlElement cityinfo in customercity)
        {
            String city = cityinfo.InnerXml;
            Console.WriteLine("\n pcid: "+ pcid +"\n city: "+city);
        }

        XmlNodeList customerprov=
selXmlDoc.SelectNodes("//cust:prov-state", nsmgr);
        foreach (XmlElement provinfo in customerprov)
        {
            String prov = provinfo.InnerXml;

```

```

        Console.WriteLine("\n pcid: "+ pcid +"\n prov-state: "+prov);
    }

    XmlNodeList customerzip= selXmlDoc.SelectNodes("//cust:pcode-zip",
nsmgr);
    foreach (XmlElement zipinfo in customerzip)
    {
        String zip = zipinfo.InnerXml;
        Console.WriteLine("\n pcid: "+ pcid +"\n pcode-zip: "+zip);
    }

    XmlNodeList customerphone= selXmlDoc.SelectNodes("//cust:phone",
nsmgr);
    foreach (XmlElement phoneinfo in customerphone)
    {
        String ptype= phoneinfo.GetAttribute("type");
        String phone = phoneinfo.InnerXml;
        Console.WriteLine("\n pcid: "+ pcid +"\n phonetype: "+ptype+",
phone: "+phone);
    }
}catch (Exception e) {
    Console.WriteLine("\nInvalid cid has been specified for the
select.\n");
    Console.WriteLine(e.Message);
}finally {
    xr.Close();
    conn.Close();
}
}
}

```

Inserting new customer data

Inserting new Customer Information requires creation of data for CID (customer ID) and INFO (customer information). The HISTORY column information is not required.

Since CID is the primary key and thus needs to be unique, we issue a simple query against the CUSTOMER table's CID column to obtain the current MAX value, then add one to generate the new CID.

For the primary key CID generation, we used MAX()+1 but this is not ideal way to generate the key. You should utilize identity column, which provides a method for DB2 database to automatically generate a unique numeric value for each row in a table.

An identity column that is defined as generated always prevents the overriding of values in an SQL statement. An identity column that is defined as generated by default gives an application a way to explicitly provide a value for the identity column.

In this case, Sample CUSTOMER table was not created with identity column.

The creation of INFO XML data is done by a separate CreateCustXML() method as shown in Example 6-12 on page 290.

The Example 6-11 shows insert of new customer CID and INFO into the CUSTOMER table. Once XmlDocument object containing customer info data has been created by CreateCustXML() method, we can now insert the data into the table.

The CID value can be concatenated to the CommandText string. However, the XmlDocument data needs to be first added using command.Parameters.Add() method after it is converted to String using InnerXml method. Note that @ prefix for the parameter is required:

```
cmd.Parameters.Add(new DB2Parameter("@XMLData", XmlDocument.InnerXml));
```

Example 6-11 Inserting new CID and info data into the CUSTOMER table

```
public static void Main(String[] args)
{
    ...
    DB2Connection conn = null;
    conn = ConnectDb(alias, userid, password);
    ...
    InsertCustomer(conn, pname, pcountry, pstreet, pcity, pprov,
                  ppcode, ptype, pphonenum);
    ...
}

public static void InsertCustomer(DB2Connection conn, String pname,
String pcountry, String pstreet, String pcity, String pprov, String
ppcode, String ptype, String pphonenum)
{
    DB2Command cmd = conn.CreateCommand();
    XmlDocument newXmlDoc = new XmlDocument();
    try{
        cmd.CommandText = "Select MAX(cid)+1 from customer";
        long count = (long)cmd.ExecuteScalar();
        String pcid=Convert.ToString(count);
        Console.WriteLine("\n maxcid value is : " +pcid);
        newXmlDoc=CreateCustXML(pcid, pname, pcountry, pstreet, pcity,
```

```
        pprov, ppcode, ptype, pphonenumb);
    cmd.Parameters.Add(new DB2Parameter("@XMLData",
        new XmlDocument.InnerXml));
    String insertStmt="Insert into customer (cid, info)
        values (" +pcid+", @XMLdata)";
    cmd.CommandText=insertStmt;
    int rowsAffected=cmd.ExecuteNonQuery();
    Console.WriteLine("\n Inserted row(s) : "+rowsAffected+" \n");
}catch (Exception e) {
Console.WriteLine(e.Message);
}finally {
    conn.Close();
}
}
```

Creation of new customer information data for the CUSTOMER table requires creation of a new XML document. Figure 6-14 demonstrates an XML document tree for the info XML data.

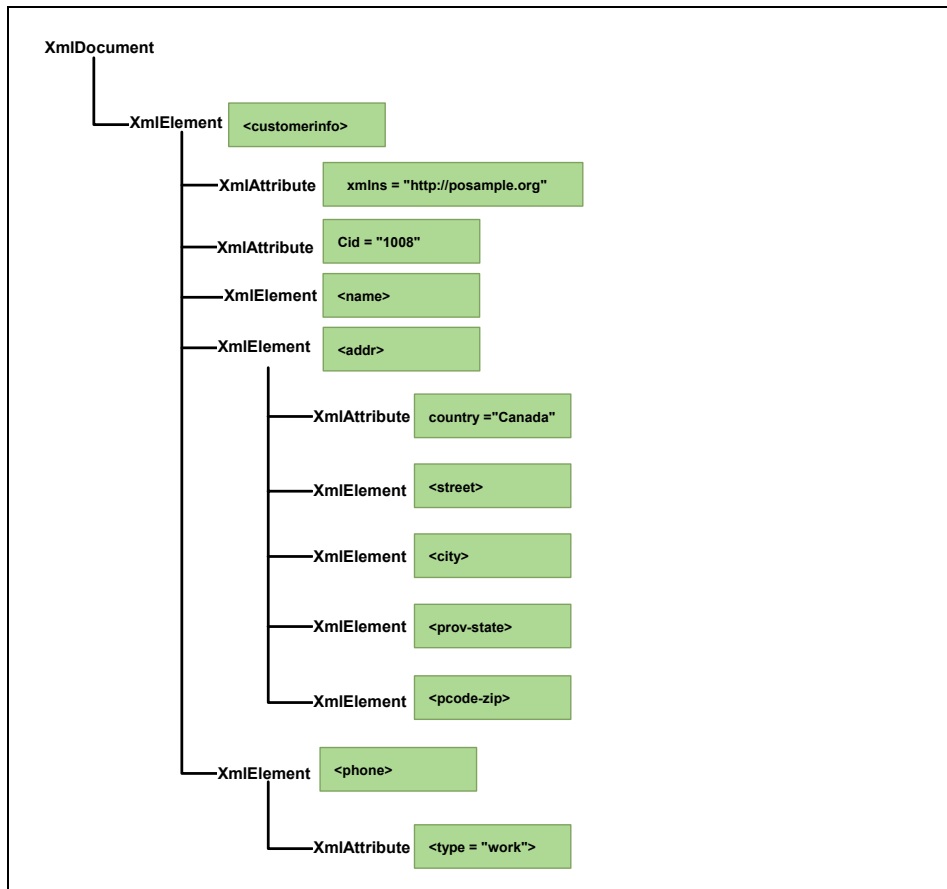


Figure 6-14 XML document tree for the customer info data

The `XmlDocument` can be created in memory using `XmlDocument()` constructor or by calling `XmlImplementation.CreateDocument()`. In our example, we create new `XmlDocument` using the constructor which will yield an empty document.

Elements will be added to the `XmlDocument` object using `CreateElement()` and `AppendChild()` methods:

```
XmlDocument.CreateElement(localname).  
XmlElement.AppendChild(Elementname);
```

Attributes can be assigned using `XmlElement.SetAttribute(localname, value)` and Element values can be assigned using `XmlElement.InnerXml = value`.

Example 6-12 Creating new info XML document

```
public static XmlDocument CreateCustXML(String icid,String iname,
String icountry, String istreet, String icity, String improv, String
ipcode, String iptype, String iphone)
{
    XmlDocument custXmlDoc = new XmlDocument();
    XmlElement customerinfo =
custXmlDoc.CreateElement("customerinfo");
    custXmlDoc.AppendChild(customerinfo);
    customerinfo.SetAttribute("xmlns", "http://posample.org");
    customerinfo.SetAttribute("Cid", icid);

    XmlElement name = custXmlDoc.CreateElement("name");
    customerinfo.AppendChild(name);
    name.InnerXml = iname;

    XmlElement addr = custXmlDoc.CreateElement("addr");
    customerinfo.AppendChild(addr);
    addr.SetAttribute("country", icountry);

    XmlElement street = custXmlDoc.CreateElement("street");
    addr.AppendChild(street);
    street.InnerXml = istreet;

    XmlElement city = custXmlDoc.CreateElement("city");
    addr.AppendChild(city);
    city.InnerXml = icity;

    XmlElement provstate = custXmlDoc.CreateElement("prov-state");
    addr.AppendChild(provstate);
    provstate.InnerXml = improv;

    XmlElement pcode = custXmlDoc.CreateElement("pcode-zip");
    addr.AppendChild(pcode);
    pcode.InnerXml = ipcode;

    XmlElement phone = custXmlDoc.CreateElement("phone");
    customerinfo.AppendChild(phone);
    phone.SetAttribute("type", iptype);
    phone.InnerXml = iphone;
```

```

        Console.WriteLine("\n XML Document to be inserted: \n " );
        custXmlDoc.Save("Customer1.xml");
        Console.WriteLine("\nDone creating customer info XML data. \n");
        return custXmlDoc;
    }

```

Updating existing customer information

Updating XML data in the CUSTOMER table will require use of the `cmd.Parameters.Add()` method to add XML data to the UPDATE statement. Customer's info XML data will be recreated using new modified information, then UPDATE statement will be used to insert new data to existing CID (customer ID). In Example 6-13, a new `XmlDocument` is created using changed/modified customer information through `CreateCustXML()` method. The `DB2Connection` object, `CID`, and `XmlDocument` object containing new info data will be passed to `UpdateCustomer()` method for update.

Example 6-13 Updating the Customer table

```

public static void Main(String[] args)
{
    ...
    int rowsupdated= 0;
    XmlDocument newXmlDoc = new XmlDocument();
    // Declare a DB2Connection
    DB2Connection conn = null;
    conn = ConnectDb(alias, userid, password);
    newXmlDoc=CreateCustXML(pcid, pname, pcountry, pstreet, pcity,
pprov, ppcode, ptype, pphonenumb);
    ...
    rowsupdated=UpdateCustomer(conn, cid, newXmlDoc);
    Console.WriteLine("\nRows updated : "+rowsupdated);
    ...
}

public static int UpdateCustomer(DB2Connection conn, int pcid,
XmlDocument pinfo)
{
    DB2Command cmd = conn.CreateCommand();
    cmd.Parameters.Add(new DB2Parameter("@XMLData", pinfo.InnerXml));
    String updtStmt="UPDATE customer SET cid = " + pcid +
" ,info = @XMLdata where cid ="+pcid+";" ;
    Console.WriteLine("\nUpdate Stmt: \n"+updtStmt);
    cmd.CommandText= updtStmt;
    int rowsUpdated = cmd.ExecuteNonQuery();
}

```

```

        Console.WriteLine("\nRows Updated: \n"+rowsUpdated);
        conn.Close();
        return rowsUpdated;
    }

```

If we need to update customer information using the name of the customer, we need to first obtain the customer ID (CID). The CID is an attribute of customerinfo element in the info XML data, which needs to be queried using the customer name.

The `GetCustomerID()` method shown in Example 6-15 will demonstrate how `XMLQuery` can be used to obtain the CID attribute using the name element.

The `XMLQuery` that will yield CID using the customer name is shown in Example 6-14.

Example 6-14 XMLQuery yielding CID

```

SELECT XMLCAST(xmlquery('declare default element namespace
"http://posample.org";$d/customerinfo/@Cid' passing customer.info as
"d" ) AS int) FROM customer where xmlexists('declare default element
namespace "http://posample.org" ;$d/customerinfo[name= "Customer Name"
]' passing customer.info as "d")

```

Example 6-15 GetCustomerID method which will retrieve CID using XMLQuery

```

public static void GetCustomerID(DB2Connection conn, String pname)
{
    int cid=0;
    String xmlquerystmt= "SELECT XMLCAST(xmlquery('declare default
element namespace "+
    "\"http://posample.org\";$d/customerinfo/@Cid' passing customer.info
as \"d\")"+
    "AS int) FROM customer where xmlexists('declare default element
namespace "+
    "\"http://posample.org\";$d/customerinfo[name=\""+pname+"\"]'
passing customer.info as \"d\");";
    Console.WriteLine("XMLQuery varchar(10) : "+xmlquerystmt);
    DB2Command cmd = new DB2Command(xmlquerystmt, conn);
    DB2DataReader reader = cmd.ExecuteReader();
    try
    {
        while (reader.Read())
        {
            //Read in the int value and assign to cid
            cid=reader.GetInt32(0);
        }
    }
}

```

```

        //Print value
        Console.WriteLine("\nCustomer "+pname+" has cid "+cid);
    }
} catch (Exception e)
{
    Console.WriteLine(e.Message);
} finally
{
    //Clean up the resources
    reader.Close();
    conn.Close();
}
}

```

Deleting customer entry

Deleting existing customer entry is done simply using DELETE statement with WHERE clause for CID as shown in Example 6-16.

Example 6-16 Deleting a row from the Customer table

```

public static void Main(String[] args)
{
    ...

    DB2Connection conn = null;
    conn = ConnectDb(alias, userid, password);
    ...
    rowsdeleted=DeleteCustomer(conn, cid);
    Console.WriteLine("\nRows deleted : "+rowsdeleted);
    ...
}

public static int DeleteCustomer (DB2Connection conn, int pcid)
{
    String delSQL="delete from customer where cid = "+pcid;
    Console.WriteLine("\nDelete Stmt: \n"+delSQL);
    DB2Command cmd = new DB2Command(delSQL, conn);
    int rowsDeleted = cmd.ExecuteNonQuery();
    Console.WriteLine("\nRows Deleted: \n"+rowsDeleted);
    conn.Close();
    return rowsDeleted;
}

```



Setup procedure and sample data

This appendix provides the following:

- ▶ Complete XML data used in the examples used throughout Chapter 2, “Application development with DB2 pureXML” on page 49.

We begin by reproducing the movies XML data, showing complete contents for five movies. Then we present movie review XML data for these five movies. Finally, we describe the simple steps to insert this XML data into the tables. The data files `movie.xml` and `moviereview.xml` and the DB2 script to set up the database are also available to download from the following IBM Redbooks Web site:

<ftp://www.redbooks.ibm.com/redbooks/SG247301>

For the download details, refer to Appendix C, “Additional material” on page 319.

- ▶ The procedure to set up Apache HTTP server, PHP, and DB2 on Windows. Apache HTTP server, PHP, and DB2 are used in the scenario in Chapter 3, “Application development with PHP” on page 93.

A.1 Example data

Example A-1 shows the movie XML data, with a total of five movies in the XML.

Example: A-1 Movie.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<movie id = "123">
  <heading>
    <title>Peterson's New Life </title>
    <rating>****</rating>
  </heading>
  <movie-details>
    <genres>Drama, Romance and Remake</genres>
    <MPAArating>PG13</MPAArating>
    <country>US</country>
    <year>2006</year>
    <running-time>1 hr. 49 minutes</running-time>
    <production>
      <studio>JJ Studio</studio>
    </production>
  </movie-details>

  <synopsis>
    After losing their 12-year-old son, Steve and Maggie Peterson moved to
    a small town. The unexpected visiting of the granddaughter from the
    next door neighbor has brought them surprise after surprise.
  </synopsis>
  <credits>
    <writer>Michael Sileur</writer>
    <director>John and Anne Tate</director>
    <photography>Charles Govasky</photography>
    <actors>
      <actor special="yes" type="lead" gender="M">Sam Wilson</actor>
      <actor type="lead" gender="F">Kate Henderson</actor>
      <actor type="lead" gender="M" special="yes">Michael Glaski</actor>
      <actor gender="M">Matt Warrich</actor>
      <actor gender="M">Chris Rogen</actor>
      <actor gender="F">Amanda Johnson</actor>
      <actor gender="M">Bill Hagen</actor>
    </actors>
  </credits>
</movie>

<?xml version="1.0" encoding="UTF-8"?>
<movie id = "234">
  <heading>
    <title>Paradise Regained</title>
    <rating>*****</rating>
```



```

</heading>
  <movie-details>
    <genres>Drama, Romance and Remake</genres>
    <MPAArating>PG13</MPAArating>
      <country>US</country>
      <year>2006</year>
      <running-time>1 hr. 54 minutes</running-time>
      <production>
        <studio>Wen International</studio>
      </production>
    </movie-details>

<synopsis>
  A down on his luck actor who never had the good fortune of getting
  the ideal part pulls off of the freeway and arrives in a sleepy little
  town of Paradise Lost.
</synopsis>
<credits>
  <writer>John Lass, Joe Farnete and Jorgen Lubien</writer>
  <director>John Lass and Joe Farnete</director>
  <photography>Charles Govasky</photography>
  <actors>
    <actor special="yes" type="lead" gender="M">Voice of Tom
Wilson</actor>
    <actor type="lead" gender="F">Voice of Kathy Hunt</actor>
    <actor type="lead" gender="M" special="yes">Larry Cable</actor>
    <actor gender="M">John Ratzenr</actor>
    <actor gender="M">Carl Wallis (II)</actor>
  </actors>
</credits>
</movie>

<?xml version="1.0" encoding="UTF-8"?>
<movie id = "345">
  <heading>
    <title>Crossroads</title>
    <rating>***</rating>
  </heading>
  <movie-details>
    <genres>Drama, Romance and Remake</genres>
    <MPAArating>PG</MPAArating>
      <country>US</country>
      <year>2006</year>
      <running-time>1 hr. 38 minutes</running-time>
      <production>
        <studio>RR Pictures</studio>
      </production>
    </movie-details>

```

```

<synopsis>
  A grouchy old college professor who lived in a dilapidated cabin near
  the lake takes an interest in a frustrated writer who suffers from
  habitual writer's block.
</synopsis>
<credits>
  <writer>David Anderson</writer>
  <director>Alex Ostern</director>
  <photography>Charles Govasky</photography>
  <actors>
    <actor special="yes" type="lead" gender="M">Karl Thomas</actor>
    <actor type="lead" gender="F">Sandra Casper</actor>
    <actor type="lead" gender="M" special="yes">Peter Walsh</actor>
    <actor gender="M">Sam Allen</actor>
    <actor gender="M">Chris Plummer</actor>
    <actor gender="F">Linda Collins</actor>
    <actor gender="M">Bill Hagen</actor>
  </actors>
</credits>
</movie>'

```

```

<?xml version="1.0" encoding="UTF-8"?>
<movie id = "456">
  <heading>
    <title>The Royal Marriage</title>
    <rating>*****</rating>
  </heading>
  <movie-details>
    <genres>Comedy</genres>
    <MPAArating>PG</MPAArating>
    <country>US</country>
    <year>1987</year>
    <running-time>1 hr. 40 minutes</running-time>
    <production>
      <studio>Century Production</studio>
    </production>
  </movie-details>

```

```

<synopsis>
  A fairy tale about a beautiful girl who married a prince and
  discovered what the true meaning of love is in a royal marriage.
</synopsis>
<credits>
  <writer>Bill Gulden</writer>
  <director>Rob Reiner</director>
  <photography>Charles Minsky</photography>
  <actors>
    <actor special="yes" type="lead" gender="M">Cary Lewies</actor>
    <actor type="lead" gender="F">Mandy Ramsey</actor>

```

```

        <actor type="lead" gender="M" special="yes">Chris Guest</actor>
        <actor gender="M">Eric Shawn</actor>
        <actor gender="M">Roger Rogen</actor>
        <actor gender="F">Amanda Johnson</actor>
        <actor gender="M">Bill Hagen</actor>
    </actors>
</credits>
</movie>

<?xml version="1.0" encoding="UTF-8"?>
<movie id = "567">
    <heading>
        <title>As Luck Would Have It</title>
        <rating>****</rating>
    </heading>
    <movie-details>
        <genres>Drama, Romance and Remake</genres>
        <MPAArating>PG</MPAArating>
        <country>US</country>
        <year>1996</year>
        <running-time>1 hr. 27 minutes</running-time>
        <production>
            <studio>RR Pictures</studio>
        </production>
    </movie-details>

    <synopsis>
        The never-do-well inventor inspected his garage full of useless
        inventions when he heard a loud knock at the front door. Upon answering
        the door, a man dressed in a conservative blue suit starts to make
        suspicious inquiries about his patents and inventions.
    </synopsis>
    <credits>
        <writer>Tim Harris</writer>
        <director>Joe Peterson</director>
        <photography>Alex Minsky</photography>
        <actors>
            <actor special="yes" type="lead" gender="M">Michael Fox</actor>
            <actor type="lead" gender="F">Terry Randy</actor>
            <actor type="lead" gender="M" special="yes">Gerry Knight</actor>
            <actor gender="M">Bill Nelson</actor>
            <actor gender="M">Charles Berkeley</actor>
            <actor gender="F">Amanda Johnson</actor>
            <actor gender="M">Bill Hagen</actor>
        </actors>
    </credits>
</movie>

```

Example A-2 shows the movie review XML.

Example: A-2 Moviereview.xml

```
<movie id = "123">
  <reviews>
    <UserReview>
      <user name ="Andy">Three is a crowd, yet there are not enough stars
to support this catastrophe.</user>
      <user name ="Linda">has half the violence but twice the laughs of the
first film</user>
    </UserReview>
    <CriticsReview>
      <newspaper name="ABC Times">The best mystery thriller of the year.
</newspaper>
      <newspaper name="San Francisco Post">a gritty, ultra-dark thriller
</newspaper>
    </CriticsReview>
  </reviews>
</movie>
<movie id = "234">
  <reviews>
    <UserReview>
      <user name ="Andy">May I start of by saying this is the first "A" I
have ever given.</user>
      <user name ="Linda">Three is a crowd, yet there are not enough stars
to support this catastrophe.</user>
    </UserReview>
    <CriticsReview>
      <newspaper name="ABC Times">The best mystery thriller of the year.
</newspaper>
      <newspaper name="San Francisco Post">The best mystery thriller of the
year.</newspaper>
    </CriticsReview>
  </reviews>
</movie>
<movie id = "345">
  <reviews>
    <UserReview>
      <user name ="Andy">I tried so hard to like this movie...I really did.
But no matter which way I look at it, I can't seem to find it in me to give
this movie a good rating.</user>
      <user name ="Linda">Three is a crowd, yet there are not enough stars
to support this catastrophe.</user>
    </UserReview>
    <CriticsReview>
      <newspaper name="ABC Times">The best mystery thriller of the
year.</newspaper>
```

```

        <newspaper name="San Francisco Post">The best mystery thriller of the
year.</newspaper>
    </CriticsReview>
</reviews>
</movie>
<movie id = "456">
    <reviews>
        <UserReview>
            <user name ="Andy">Three is a crowd, yet there are not enough stars
to support this catastrophe.</user>
            <user name ="Linda">Almost jokeless and at times nearly
incomprehensible...</user>
        </UserReview>
        <CriticsReview>
            <newspaper name="ABC Times">The best mystery thriller of the year.
</newspaper>
            <newspaper name="San Francisco Post">The best mystery thriller of the
year. </newspaper>
        </CriticsReview>
    </reviews>
</movie>
<movie id = "567">
    <reviews>
        <UserReview>
            <user name ="Andy">Three is a crowd, yet there are not enough stars
to support this catastrophe.</user>
            <user name ="Linda">Well here goes! We caught the advance screening
here</user>
        </UserReview>
        <CriticsReview>
            <newspaper name="ABC Times">The best mystery thriller of the
year.</newspaper>
            <newspaper name="San Francisco Post">The best mystery thriller of the
year.</newspaper>
        </CriticsReview>
    </reviews>
</movie>

```

A.2 Setting up the database

Example A-3 on page 302 shows the script to set up the database for the Movie review example. This is the only script needed to be run to set up the database used in the example throughout Chapter 2, “Application development with DB2 pureXML” on page 49. Run the script `movies.db2` as below:

```
db2 -td@ -vf movies.db2
```

Example: A-3 movies.db2 (set up database script)

```
create database movie using codeset utf-8 territory us@
connect to movie@
drop table movies@
create table movies(id int not null primary key, info XML)@
drop table moviereview@
create table moviereview(reviewid int not null primary key, review xml)@

insert into moviereview(reviewid, review) values(321, xmlparse( document
'<movie id = "123">
  <reviews>
    <UserReview>
      <user name ="Andy">Three is a crowd, yet there are not enough stars
to support this catastrophe.</user>
      <user name ="Linda">has half the violence but twice the laughs of the
first film</user>
    </UserReview>
    <CriticsReview>
      <newspaper name="ABC Times">The best mystery thriller of the year.
</newspaper>
      <newspaper name="SF Post">a gritty, ultra-dark thriller </newspaper>
    </CriticsReview>
  </reviews>
</movie>'))@

insert into moviereview(reviewid, review) values(432, xmlparse( document
'<movie id = "234">
  <reviews>
    <UserReview>
      <user name ="Andy">May I start of by saying this is the first "A" I
have ever given.</user>
      <user name ="Linda">Three is a crowd, yet there are not enough stars
to support this catastrophe.</user>
    </UserReview>
    <CriticsReview>
      <newspaper name="ABC Times">The best mystery thriller of the year.
</newspaper>
      <newspaper name="SF Post">The best mystery thriller of the
year.</newspaper>
    </CriticsReview>
  </reviews>
</movie>'))@

insert into moviereview(reviewid, review) values(543, xmlparse( document
'<movie id = "345">
  <reviews>
    <UserReview>
```

```

        <user name ="Andy">I tried so hard to like this movie...I really did.
        But no matter which way I look at it, I can't seem to find it in me to give
        this movie a good rating.</user>
        <user name ="Linda">Three is a crowd, yet there are not enough stars
        to support this catastrophe.</user>
        </UserReview>
        <CriticsReview>
        <newspaper name="ABC Times">The best mystery thriller of the
        year.</newspaper>
        <newspaper name="SF Post">The best mystery thriller of the
        year.</newspaper>
        </CriticsReview>
    </reviews>
</movie>'))@

```

```

insert into moviereview(reviewid, review) values(654, xmlparse( document
'<movie id = "456">
  <reviews>
    <UserReview>
      <user name ="Andy">Three is a crowd, yet there are not enough stars
      to support this catastrophe.</user>
      <user name ="Linda">Almost jokeless and at times nearly
      incomprehensible...</user>
    </UserReview>
    <CriticsReview>
      <newspaper name="ABC Times">The best mystery thriller of the year.
    </newspaper>
      <newspaper name="SF Post">The best mystery thriller of the year.
    </newspaper>
    </CriticsReview>
  </reviews>
</movie>'))@

```

```

insert into moviereview(reviewid, review) values(765, xmlparse( document
'<movie id = "567">
  <reviews>
    <UserReview>
      <user name ="Andy">Three is a crowd, yet there are not enough stars
      to support this catastrophe.</user>
      <user name ="Linda">Well here goes! We caught the advance screening
      here</user>
    </UserReview>
    <CriticsReview>
      <newspaper name="ABC Times">The best mystery thriller of the
      year.</newspaper>
      <newspaper name="SF Post">The best mystery thriller of the
      year.</newspaper>
    </CriticsReview>
  </reviews>

```

```
</movie>'))@
```

```
insert into movies(id, info) values(123, xmlparse(document '<?xml version="1.0"
encoding="UTF-8"?>
```

```
<movie id = "123">
```

```
<heading>
```

```
<title>Peterson's New Life </title>
```

```
<rating>****</rating>
```

```
</heading>
```

```
<movie-details>
```

```
<genres>Drama, Romance and Remake</genres>
```

```
<MPAArating>PG13</MPAArating>
```

```
<country>US</country>
```

```
<year>2006</year>
```

```
<running-time>1 hr. 49 minutes</running-time>
```

```
<production>
```

```
<studio>JJ Studio</studio>
```

```
</production>
```

```
</movie-details>
```

```
<synopsis>
```

```
After losing their 12-year-old son, Steve and Maggie Peterson moved to
a small town. The unexpected visiting of the granddaughter from the
next door neighbor has brought them surprise after surprise.
```

```
</synopsis>
```

```
<credits>
```

```
<writer>Michael Sileur</writer>
```

```
<director>John and Anne Tate</director>
```

```
<photography>Charles Govasky</photography>
```

```
<actors>
```

```
<actor special="yes" type="lead" gender="M">Sam Wilson</actor>
```

```
<actor type="lead" gender="F">Kate Henderson</actor>
```

```
<actor type="lead" gender="M" special="yes">Michael Glaski</actor>
```

```
<actor gender="M">Matt Warrich</actor>
```

```
<actor gender="M">Chris Rogen</actor>
```

```
<actor gender="F">Amanda Johnson</actor>
```

```
<actor gender="M">Bill Hagen</actor>
```

```
</actors>
```

```
</credits>
```

```
</movie>'))@
```

```
insert into movies(id, info) values(234, xmlparse(document '<?xml version="1.0"
encoding="UTF-8"?>
```

```
<movie id = "234">
```

```
<heading>
```

```
<title>Paradise Regained</title>
```

```
<rating>*****</rating>
```

```
</heading>
```



```

<movie-details>
<genres>Drama, Romance and Remake</genres>
<MPAArating>PG13</MPAArating>
  <country>US</country>
  <year>2006</year>
  <running-time>1 hr. 54 minutes</running-time>
  <production>
    <studio>Wen International</studio>
  </production>
</movie-details>

<synopsis>
  A down on his luck actor who never had the good fortune of getting
  the ideal part pulls off of the freeway and arrives in a sleepy little
  town of Paradise Lost.
</synopsis>
<credits>
  <writer>John Lass, Joe Farnete and Jorgen Lubien</writer>
  <director>John Lass and Joe Farnete</director>
  <photography>Charles Govasky</photography>
  <actors>
    <actor special="yes" type="lead" gender="M">Voice of Tom
Wilson</actor>
    <actor type="lead" gender="F">Voice of Kathy Hunt</actor>
    <actor type="lead" gender="M" special="yes">Larry Cable</actor>
    <actor gender="M">John Ratzenr</actor>
    <actor gender="M">Carl Wallis (II)</actor>

  </actors>
</credits>
</movie>'))@

insert into movies(id, info) values(345, xmlparse(document '<?xml version="1.0"
encoding="UTF-8"?>
<movie id = "345">
  <heading>
    <title>Crossroads</title>
    <rating>***</rating>
  </heading>
  <movie-details>
  <genres>Drama, Romance and Remake</genres>
  <MPAArating>PG</MPAArating>
  <country>US</country>
  <year>2006</year>
  <running-time>1 hr. 38 minutes</running-time>
  <production>
    <studio>RR Pictures</studio>
  </production>
</movie-details>

```

```

<synopsis>
  A grouchy old college professor who lived in a dilapidated cabin near
  the lake takes an interest in a frustrated writer who suffers from
  habitual writer's block.
</synopsis>
<credits>
  <writer>David Anderson</writer>
  <director>Alex Ostern</director>
  <photography>Charles Govasky</photography>
  <actors>
    <actor special="yes" type="lead" gender="M">Karl Thomas</actor>
    <actor type="lead" gender="F">Sandra Casper</actor>
    <actor type="lead" gender="M" special="yes">Peter Walsh</actor>
    <actor gender="M">Sam Allen</actor>
    <actor gender="M">Chris Plummer</actor>
    <actor gender="F">Linda Collins</actor>
    <actor gender="M">Bill Hagen</actor>
  </actors>
</credits>
</movie>'))@

```

```

insert into movies(id, info) values(456, xmlparse(document '<?xml version="1.0"
encoding="UTF-8"?>

```

```

<movie id = "456">
  <heading>
    <title>The Royal Marriage</title>
    <rating>*****</rating>
  </heading>
  <movie-details>
    <genres>Comedy</genres>
    <MPAArating>PG</MPAArating>
    <country>US</country>
    <year>1987</year>
    <running-time>1 hr. 40 minutes</running-time>
    <production>
      <studio>Century Production</studio>
    </production>
  </movie-details>

```

```

<synopsis>
  A fairy tale about a beautiful girl who married a prince and
  discovered what the true meaning of love is in a royal marriage.
</synopsis>
<credits>
  <writer>Bill Gulden</writer>
  <director>Rob Reiner</director>
  <photography>Charles Minsky</photography>
  <actors>

```

```

        <actor special="yes" type="lead" gender="M">Cary Lewies</actor>
        <actor type="lead" gender="F">Mandy Ramsey</actor>
        <actor type="lead" gender="M" special="yes">Chris Guest</actor>
        <actor gender="M">Eric Shawn</actor>
        <actor gender="M">Roger Rogen</actor>
        <actor gender="F">Amanda Johnson</actor>
        <actor gender="M">Bill Hagen</actor>
    </actors>
</credits>
</movie>'))@

```

```

insert into movies(id, info) values(567, xmlparse(document '<?xml version="1.0"
encoding="UTF-8"?>

```

```

<movie id = "567">
    <heading>
        <title>As Luck Would Have It</title>
        <rating>****</rating>
    </heading>
    <movie-details>
        <genres>Drama, Romance and Remake</genres>
        <MPAArating>PG</MPAArating>
        <country>US</country>
        <year>1996</year>
        <running-time>1 hr. 27 minutes</running-time>
        <production>
            <studio>RR Pictures</studio>
        </production>
    </movie-details>

    <synopsis>
        The never-do-well inventor inspected his garage full of useless
        inventions when he heard a loud knock at the front door. Upon answering
        the door, a man dressed in a conservative blue suit starts to make
        suspicious inquiries about his patents and inventions.
    </synopsis>
    <credits>
        <writer>Tim Harris</writer>
        <director>Joe Peterson</director>
        <photography>Alex Minsky</photography>
        <actors>
            <actor special="yes" type="lead" gender="M">Michael Fox</actor>
            <actor type="lead" gender="F">Terry Randy</actor>
            <actor type="lead" gender="M" special="yes">Gerry Knight</actor>
            <actor gender="M">Bill Nelson</actor>
            <actor gender="M">Charles Berkeley</actor>
            <actor gender="F">Amanda Johnson</actor>
            <actor gender="M">Bill Hagen</actor>
        </actors>
    </credits>

```

A.3 Setting up Apache HTTP server, PHP, and DB2 on Windows

We understand that it is sometimes frustrating to set up these technologies to work together, specially for the first time user. Hence, we decided to provide you some basic steps to install, configure, and run a quick test. These steps have been tested on Windows XP Professional, but should work on other Windows operating systems. The same steps can be utilized with little modification on other operating systems such as Linux as well.

Install Apache HTTP Server

Follow these steps to install Apache HTTP server:

- ▶ Install the latest stable version of Windows binary installer (file name must end with msi) from the following Web site:
<http://httpd.apache.org/download.cgi>
- ▶ Follow the prompt of the installer carefully and finish the install. This should be installed by default in the directory as below:
C:\Program Files\Apache Group\Apache2
- ▶ Point your browser to the URL below, you should be able to see the Apache HTTP Server Index page in English.
<http://localhost/>
- ▶ In case of a problem, go to **Start** → **All Programs** → **Apache** and try to stop and start or restart. The "Monitor Apache Servers" utility may also appear in your Windows Taskbar System Tray.

Install and configure PHP

Use the following steps to install and configure PHP:

- ▶ Download the latest stable PHP zip package for Windows (currently 5.1.5 available) from the Web site below. Caution, do not choose Windows installer because it is CGI only.
<http://www.php.net/downloads.php>
- ▶ Unzip it in C:\PHP folder.
- ▶ Download the collection of PECL modules from the same Web site as mentioned above. The PECL package contains the extensions and libraries that are not part of core PHP libraries. The IBM DB2 driver for PHP comes in the PECL package.

- ▶ Unzip the PECL packages in C:\PHP\ext folder.
- ▶ Edit the Apache HTTP server configuration file, so that the Web server recognizes PHP. Open C:\Program Files\Apache Group\Apache2\conf\httpd.conf file in your favorite editor and add the following two lines:

```
LoadModule php5_module C:/php/php5apache2.d11
AddType application/x-httpd-php .php
```

You can add anywhere in the httpd.conf file, but we recommend you find the corresponding section for LoadModule and AddType and add them at the bottom of that section.

- ▶ Copy and paste the file C:\PHP\php.ini-dist to C:\windows directory and rename it as php.ini.
- ▶ Restart the Apache HTTP server and test PHP by creating a simple text file called info.php with the contents below:

```
<?phpinfo();?>
```

- ▶ Point your Web browser to the URL below. You should be able to see the complete PHP setting/configuration Web page.

```
http://localhost/info.php
```

- ▶ (Optional) In some cases, you might need to copy the following PHP libraries from C:\PHP directory to your Apache directory C:\Program Files\Apache Group\Apache2:

```
php5apache2.d11
php5apache.d11
phpapache_hooks.d11
php5isapi.d11
php5nsapi.d11
php5ts.d11
```

Install and configure DB2 extension

Follow these steps to install and configure DB2 extension:

- ▶ Edit the C:\windows\php.ini file. Add the IBM DB2 extension below:


```
extension=php_ibm_db2.d11
```
- ▶ Create a test file as shown in Example 6-17 to test the PHP with DB2. This assumes that you have DB2 V9 Express-C installed on your computer and that SAMPLE database exists. This test program verifies that you have the latest DB2 driver installed, which can leverage new pureXML functionalities discussed in the redbook.

Example 6-17 sample PHP code to test DB2

```
<?php
$conn = db2_connect('sample', 'userid', 'password');
$create = 'DROP TABLE xmlTest';
$result = db2_exec($conn, $create);
$create = 'CREATE TABLE xmlTest (id INTEGER, data XML)';
$result = db2_exec($conn, $create);
$insert = "INSERT INTO xmlTest values (0,
'<Client><Address><street>555 Bailey Ave</street><city>San
Jose</city><state>CA</state><zip>95141</zip></Address><email>ranjanr
@us.ibm.com</email></Client>')";

db2_exec( $conn, $insert );
if ($conn) {
    $sql = "SELECT data FROM xmlTest";
    $stmt = db2_prepare( $conn, $sql );
    db2_execute($stmt);
    while($result = db2_fetch_assoc($stmt)) {
        print_r($result);
        echo "\n";
    }
    db2_close($conn);
} else {
    echo "Connection failed.\n";
}
?>
```

- ▶ Save this file in any folder(for example, C:\rakesh\test.php) and run it:
php test.php
- ▶ You should be able to see the result as shown in Example 6-18 on page 310.

Example 6-18 Output of test.php

```
C:\rakesh>php test.php
Array
(
    [DATA] => <?xml version="1.0" encoding="UTF-8"
?><Client><Address><street>555 Bailey Ave</street><city>San
Jose</cit
y><state>CA</state><zip>95141</zip></Address><email>ranjanr@us.ibm.c
om</email></Client>
)
```

Note that PHP can also be used on the command line. This method of running PHP is called PHP CLI. Once you have PHP installed and configured, you can test it by running the **php -v** command on your command prompt window. This should tell you which version of PHP is running, similar to output shown in Example 6-19:

Example 6-19 php -v

```
C:\rakesh>php -v
PHP 5.1.4 (cli) (built: May 4 2006 10:35:22)
Copyright (c) 1997-2006 The PHP Group
Zend Engine v2.1.0, Copyright (c) 1998-2006 Zend Technologies
```

That is it! You have now installed the finest tools to create database driven Web applications. Enjoy!

Ruby on Rails

This appendix briefly introduces Ruby and Ruby on Rails. This appendix includes the following topics:

- ▶ Introduction to Ruby
- ▶ Introduction to Ruby on Rails

B.1 Introduction to Ruby

Ruby is a object-oriented programming language. It was created by Yukihiro "Matz" Matsumoto in 1995. Ruby language has evolved since then and at the time of writing this book, the latest stable version of Ruby is 1.8.4. Ruby's syntax was inspired by Perl and Smalltalk and also shares some features with Python and LISP. Ruby is available as a free software download from its official Web site given below. This Web site also provides documentation and a community mailing list:

<http://www.ruby-lang.org/en/20020102.html>

Ruby language is well documented (as of writing this book there are more than 9000 methods that are available in Ruby language). you can find both core API and standard library documentation on the following Web site:

<http://ruby-doc.org/>

In today's software development world, we have choices. Ruby can be considered to be a great choice for enterprise Web application development because it distinguishes itself from other available choices in many ways. In order to understand these features, you would need to refer to the Ruby language documentation, mentioning that in detail is beyond the scope of this appendix.

- ▶ Ruby employs a clean syntax. You are not required to put a semicolon at the end of the statement. It can have disadvantages also, because a programmer has to take care of code layout independently.
- ▶ Ruby has regular expressions built into the language. They are also treated as objects and manipulated in the program for example, pattern matching, and so on.
- ▶ Ruby is a single inheritance only, that means a Ruby class can have only one parent. However, Ruby provides multiple inheritance-type functionality by allowing classes to include the functionalities from a partial class definition.
- ▶ Ruby provides a hierarchy of exception classes, which is easy to use.
- ▶ Ruby implements threads in the language interpreter code itself; hence, it is independent of the operating system. However, Ruby threads do not utilize more than one CPU.
- ▶ Ruby language comes with a built-in debugger. You can run debugger by invoking the debugger with `-r` option.
- ▶ Ruby features its own shell called "Interactive Ruby". You can simply invoke interactive Ruby by typing `irb` command on your system. This allows you to play with Ruby if you are new to the language.

- ▶ Ruby language comes with a code profiler. You can invoke it at command line or in the program.
- ▶ Ruby comes ready for Web programming. Ruby can be used to write CGI scripts for Web programming.

B.1.1 Getting started with Ruby programming language

Getting started with Ruby could not be easier. Ruby is available on all operating system platforms, including Windows. We show you how to start with Ruby. Following the new programming language learning tradition, we write and run a Hello World program in Ruby.

First, you download and install Ruby language pack from the following Web site:

http://rubyforge.org/frs/?group_id=167

Choose the latest stable version. In this example, we downloaded ruby1.8.4-20.exe and extracted the code in C:\ruby folder. Once extracted, Ruby is ready to use. Let us write our first program in Ruby.

- ▶ Change the directory to C:\ruby and edit a new notepad file with this content:

```
puts "Hello Ruby World !"
```

- ▶ Save this file as hello.rb.
- ▶ On command prompt, run it as shown:

```
C:\ruby>ruby hello.rb
```

- ▶ You should see the output as below:

```
Hello Ruby World !
```

For your learning pleasure, the Ruby package comes loaded with lots of samples. You can browse some of these samples in the following folder:

```
C:\ruby\samples\RubySrc-1.8.4\sample
```

B.2 Introduction to Ruby on Rails

Ruby on Rails (ROR) is a Web application and persistence framework that includes everything needed to create database-backed Web applications according to the Model-View-Control pattern of separation. This pattern splits the view (also called the presentation) into templates that are primarily responsible for inserting pre-built data in between HTML tags. The model contains the "smart" domain objects (such as Account, Customer, and Product) that hold all the business logic and know how to persist themselves to a database. The

controller handles the incoming requests (such as Create New Account, Update Product, and Show Post) by manipulating the model and directing data to the view. In this framework, model is handled by an object relational mapping layer called *Active Record*. Active Record connects business objects and database tables to create a persistable domain model where logic and data are presented in one wrapping. Active record also minimizes the configuration detail that a developer has to perform.

ROR comes bundled with everything, all you need is your database, and a Web server. The latest release of ROR is 1.1 and you can download it from the official ROR Web site:

<http://www.rubyonrails.org/>

B.2.1 DB2 9 on Rails

In this section, we provide you a link to a great new tool specially designed to build Web applications with Ruby on Rails and DB2. IBM has created a starter toolkit for building an application with cutting edge database technology in DB2 9. This starter toolkit is freely available on IBM alphaworks Web site and packaged with the DB2 9 database and necessary driver and adapter for Ruby. The tool can be downloaded from the Web site:

<http://alphaworks.ibm.com/tech/db2onrails>

B.2.2 Further reading

The following Web sites contains pertinent information regarding this topic:

- ▶ *Fast-track your Web apps with Ruby on Rails*

A developerworks article to develop Web-based applications using Ruby on Rails:

<http://www.ibm.com/developerworks/linux/library/l-rubyrails/>

- ▶ *Agile Web Development with Rails* by Dave Thomas

<http://www.pragmaticprogrammer.com/title/rails/>

- ▶ *Server Side programming with Ruby*

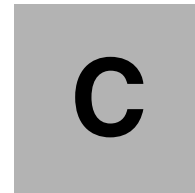
<http://www.devx.com/enterprise/Article/28101>

- ▶ *Programming Ruby - The Pragmatic Programmer's Guide*

<http://www.rubycentral.com/book/preface.html>

- ▶ *The Ruby FAQ*

<http://www.rubygarden.org/faq/main/>



Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG247301>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG247301.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
------------------	--------------------

movie.zip	Zipped Movie ranking database setup script and XML data
zframework.zip	Zipped PHP application samples - Movie ranking, Social network, and Yahoo Web service
Inventory.zip	Zipped C/C++ application code samples - Inventory
cart.zip	Zipped Java application code sample - Shopping cart
registration.zip	Zipped .Net application code sample - User registration

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	1 MB
Operating System:	Windows/Linux
Processor:	486 or higher
Memory:	512 MB

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Related publications

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

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 324. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *DB2 UDB V8.2 on the Windows Environment*, SG24-7102

Other publications

These publications are also relevant as further information sources:

IBM - DB2 9

- ▶ *What's New*, SC10-4253
- ▶ *Administration Guide: Implementation*, SC10-4221
- ▶ *Administration Guide: Planning*, SC10-4223
- ▶ *Administrative API Reference*, SC10-4231
- ▶ *Administrative SQL Routines and Views*, SC10-4293
- ▶ *Call Level Interface Guide and Reference, Volume 1*, SC10-4224
- ▶ *Call Level Interface Guide and Reference, Volume 2*, SC10-4225
- ▶ *Command Reference*, SC10-4226
- ▶ *Data Movement Utilities Guide and Reference*, SC10-4227
- ▶ *Data Recovery and High Availability Guide and Reference*, SC10-4228
- ▶ *Developing ADO.NET and OLE DB Applications*, SC10-4230
- ▶ *Developing Embedded SQL Applications*, SC10-4232
- ▶ *Developing Java Applications*, SC10-4233
- ▶ *Developing Perl and PHP Applications*, SC10-4234
- ▶ *Getting Started with Database Application Development*, C10-4252

- ▶ *Getting started with DB2 installation and administration on Linux and Windows*, GC10-4247
- ▶ *Message Reference Volume 1*, SC10-4238
- ▶ *Message Reference Volume 2*, SC10-4239
- ▶ *Migration Guide*, GC10-4237
- ▶ *Performance Guide*, SC10-4222
- ▶ *Query Patroller Administration and User's Guide*, GC10-4241
- ▶ *Quick Beginnings for DB2 Clients*, GC10-4242
- ▶ *Quick Beginnings for DB2 Servers*, GC10-4246
- ▶ *Spatial Extender and Geodetic Data Management Feature User's Guide and Reference*, SC18-9749
- ▶ *SQL Guide*, SC10-4248
- ▶ *SQL Reference, Volume 1*, SC10-4249
- ▶ *SQL Reference, Volume 2*, SC10-4250
- ▶ *System Monitor Guide and Reference*, SC10-4251
- ▶ *Troubleshooting Guide*, GC10-4240
- ▶ *Visual Explain Tutorial*, SC10-4319
- ▶ *XML Extender Administration and Programming*, SC18-9750
- ▶ *XML Guide*, SC10-4254
- ▶ *XQuery Reference*, SC18-9796
- ▶ *DB2 Connect User's Guide*, SC10-4229
- ▶ *Quick Beginnings for DB2 Connect Personal Edition*, GC10-4244
- ▶ *Quick Beginnings for DB2 Connect Servers*, GC10-4243

IBM - DB2 9

- ▶ *What's New V8*, SC09-4848-01
- ▶ *Administration Guide: Implementation V8*, SC09-4820-01
- ▶ *Administration Guide: Performance V8*, SC09-4821-01
- ▶ *Administration Guide: Planning V8*, SC09-4822-01
- ▶ *Application Development Guide: Building and Running Applications V8*, SC09-4825-01
- ▶ *Application Development Guide: Programming Client Applications V8*, SC09-4826-01

- ▶ *Application Development Guide: Programming Server Applications V8*, SC09-4827-01
- ▶ *Call Level Interface Guide and Reference, Volume 1, V8*, SC09-4849-01
- ▶ *Call Level Interface Guide and Reference, Volume 2, V8*, SC09-4850-01
- ▶ *Command Reference V8*, SC09-4828-01
- ▶ *Data Movement Utilities Guide and Reference V8*, SC09-4830-01
- ▶ *Data Recovery and High Availability Guide and Reference V8*, SC09-4831-01
- ▶ *Guide to GUI Tools for Administration and Development*, SC09-4851-01
- ▶ *Installation and Configuration Supplement V8*, GC09-4837-01
- ▶ *Quick Beginnings for DB2 Clients V8*, GC09-4832-01
- ▶ *Quick Beginnings for DB2 Servers V8*, GC09-4836-01
- ▶ *Replication and Event Publishing Guide and Reference*, SC18-7568
- ▶ *SQL Reference, Volume 1, V8*, SC09-4844-01
- ▶ *SQL Reference, Volume 2, V8*, SC09-4845-01
- ▶ *System Monitor Guide and Reference V8*, SC09-4847-01
- ▶ *Data Warehouse Center Application Integration Guide Version 8 Release 1*, SC27-1124-01
- ▶ *DB2 XML Extender Administration and Programming Guide Version 8 Release 1*, SC27-1234-01
- ▶ *Federated Systems PIC Guide Version 8 Release 1*, GC27-1224-01

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ DB2 XML wiki
<http://www.ibm.com/developerworks/wikis/display/db2xml/Home>
- ▶ DB2 Information Center
<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp>
- ▶ DB2 Express-C
<http://www.ibm.com/software/data/db2/udb/db2express/>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

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

Help from IBM

IBM Support and downloads

<http://www.ibm.com/support>

IBM Global Services

<http://www.ibm.com/services>

Index

Symbols

\$db 115
\$view 115
\$xcs 119
.c 153
.h 157
.NET 2, 11, 252, 280
? 221
@ 181, 287
__autoload 102

A

absolute() 215
access plan 153
active record 316
ActiveX Data Object 252
adapter 271
add_product() 154
Add-In 3, 253
ADO.NET 3, 252
afterLast() 215
API 153
APIs 148, 208
APP driver 8
AppendChild() 289
application 148
application programming interfaces 148
argument 4, 201
array 219
AsciiStream 224
associative array 122
atom feed 117
attribute 79
Auto commit mode 234
auto-commit attribute 234

B

beforeLast() 215
BinaryStream 224
bind file 148, 150, 153, 157, 180
BINDFILE option 151, 153
bind-in 55

bind-out 55
BLOB 58, 222, 224
bootstrap file 112
buffer 183
business logic 50
business object 316

C

C# 2, 278
C++ 150
C/C++ 2
Call Level Interface 177
CallableStatement 212
CGI 6
character string 4
character type 58
Check Constraints View 260
class 208
CLI 3, 6, 11, 148, 177–178, 180
CLI bind file 180
CLI driver 119, 180
CLOB 58, 224
COBOL 3
code page 196
codepage 91
colon 151, 156
Column Definitions View 260
com.ibm.db2.jcc 200
command
 COMMIT 163, 167
 PRECOMPILE 151
Command Line Processor 3
command.Parameters.Add() 287
CommandText 270
CommandType 270
commit 234
common section 179
communication 273
compile 160
compile time 148
components 98
CONNECT TO statement 158
ConnectDb() 280

- Connection object 225
- constructor 113
- convenience method 120
- createCart(HashMap) 201
- CreateCustXML() 287
- CreateElement() 289
- createPorder() 202
- Creating SQLJ 28
- Creating XQuery 32
- CRUD 119
- CS 236
- cursor 213
- cursor stability 236

D

- data access 2
- data consumers 267
- data model 55
- Data Provide 253
- data provider 280
- data set 271
- data source 273
- data type 4, 58, 191, 211, 223
- DataAdapter 269
- database layer 116
- database object 112, 259
- database objects 148
- database resource 2
- database transaction 233
- DatabaseMetaData 217
- DataReader 272
- DataSet 271
- DataSet object 273
- DataSource object 227
- DataTruncation 232
- DB2 .NET Data provider 11
- DB2 Developer Workbench 24
- DB2 Express-C 11
- DB2 PREP 150
- DB2 setup wizard 12–13
- db2_install 23
- db2_install script 12
- db2bfd 157
- db2cli.ini 178
- db2cli.lst 181
- DB2CLIINIPATH 178
- DB2Connection 269
- db2sampl 23

- db2setup 22
- db2sqljbind 10
- db2sqljcustomize 10
- DB2Transaction 279
- db2ubind.lst 181
- DB2Xml class 224
- DBD 6
- DBGRAPHICS 223
- DBI 6
- ddcsmvs.lst 181
- decimal 219
- declare section 156
- decomposition 65
- delimiters 4
- deployment tool 12
- DESCRIBE statement 174
- disconnect 158
- distributed transaction processing 3
- distribution 2
- document model 81
- Document Object Model 56
- DOM 56
- driver 217
- driver class 227
- DriverManager 208
- dynamic 148
- dynamic embedded SQL 4, 155
- dynamic SQL statement 148, 175

E

- e-business 2
- element 79
- embedded SQL 4, 148, 150, 154, 194
- encoding conversion 224
- enterprise application 97
- environment 94
- ErrorCode 228
- errText 161
- velopment environment 2
- exception handling 228
- EXEC SQL portion 156
- EXECUTE IMMEDIATE statement 168
- executeQuery 213
- ExecutionContext 241
- export 88, 255
- expression 66
- extensions 150

F

fetch 209
FETCH statement 174
fetch*() 113
FieldCount property 273
file extension 153
first() 215
FLWOR 44
for logic 44
forward-only 272
function 59

G

getAsciiStream() 224
getBinaryStream() 224
getBytes() 224
getCharacterStream() 224
getConnection 208
getDataSize 232
getDescription(String) 201
getIndex 232
getNextException 229
getObject() 224
getParameter 232
getParameterMetaData method 218
getProducts() 201
getRead 232
getSqlca 228
getString() 224
getThrowable 228
getTransferSize 232
GRANT 163
GRAPHICS 223

H

handle 177, 182
HandleType 183
HasRows property 273
header file 159
hierarchical form 52
holdability 245
host language 4
host variable 4, 60, 151, 156–157, 191–192

I

IBM.DATA.DB2 11
IBM.Data.DB2 278

ibm_db2 10, 119
IBMDADB2 7, 11
IEnumerable interface 284
implicit parsing 59
import 255
IN 212
indexAction() 104, 127
IndexController.php 103
Indexes View 260
INOUT 212
installation 12
isCustomer 201
isolation level 234, 236
iterators 241

J

Java 2
Java.sql 208
java.sql 200
javax.naming 200
javax.sql 200
JCC type 2 8
JCC type 4 8
JDBC 8, 200, 208
JDBC/SQLJ, 3
JNDI 227

K

kernel 2
Keys View 260
keyword 180

L

last() 215
life cycle 225
LOB 191
LOB locator 186
LOB_FILE 191

M

managed provider 267
metadata 51
method 127, 201, 252
Model-View-Control 315
Model-View-Controller 97
modular 98
monitor 3

MSDASQL 7
MVC 97

N

named iterator 241
namespace 80, 278
native structure 116
NET driver 8
Net Search Extender 81
next() 215
node 58
noRouteAction() 127
NSE 81

O

object file 151
ODBC 11
ODBC .NET Data provider 11
ODBCConnection 279
OdbcConnection 269
ODBCDataReader 279
ODBCTransaction 279
OLE DB 3, 7
OLE DB .NET Data provider 11
OleDbCommand 279
OleDbConnection 269
OleDbDataAdapter 271
OleDbTransaction 279
On Error construct 278
Open method 269
optimizer 82, 153
options 150
OS level 2
OUT 212
OutputHandlePtr 183
overhead 273

P

package 4, 150, 153, 180
parameter 60
parameter marker 188
parser 55
PDO_ODBC 10
performance 79
Perl 3, 6
Perl DBI 3
persistence data access 117

PHP 2, 10
placeholder 151
port 111
portability 95
positioned iterator 241
PRECOMPILE 150
precompile 4, 150
precompile time 151
precompiler 150, 153
PREPARE 167
PreparedStatement interface 212
PreparedStatement object 212, 218
presentation 315
previous() 215
privilege 264
Privileges View 260, 262, 264
procedure parameter 264
Procedure View 264
procedures 259
product family 2
programming interface 2
programming language 4
protocol 50, 53
pure XML 94
pureXML 52
pureXML storage 116

Q

query 213
query_update() 154
question marker (?) 151

R

read stability 236
read-only 272
record set 272
Redbooks Web site 324
 Contact us xiv
registry 178
registry variable 115
relative() 215
repeatable read 236
repository 118
response file 22
response file install 12
RESS 3
result set 163, 209
ResultSet 209

ResultSet object 213
resultSetConcurrency 214
resultSetHoldability 214
resultSetType 214
ROLLBACK 163
ROR 315
RR 236
RS 236
RSS feed 117
runtime 148, 167

S

Sample database 23
savepoint 237, 244
scalar function search 82
schema 60
schema document 60
schema validation 55
scripting language 97
scripts 259
scrollability 214, 245
SelectCustomer() 284
semicolon 156, 179
serialized file 4
serialized string 192
setAsciiStream() 223
setBinaryStream() 223
setBlob() 224
setBytes() 223
setCharacterStream() 224
setClob() 224
setObject() 224
setString() 223
setup 22
setXXX method 211
Show Script View 260, 262, 264
shredding 65
source file 150, 153, 159
SQL 3, 94
SQL Communications Area 159
SQL Descriptor Area 174
SQL statement 150
sql.h 161
SQL_C_BINARY 196
SQL_HANDLE_DBC 177
SQL_HANDLE_DESC 177
SQL_HANDLE_ENV 177
SQL_HANDLE_STMT 177

SQL_XML 196
SQLAllocHandle() 183
SQLBindCol() 186
SQLBrowseConnect 185
SQLCA 159, 161
SQLCA structure 161
Sqlcli.h 183
Sqlcli1.h 183
SQLCODE 159
SQLColAttribute 187
SQLConnect 185
SQLConnect() 185
sqld 176
SQLDisconnect 185
SQLDriverConnect 185
SQLDriverConnect() 185
SQLERRD 159
SQLException 228, 232
SQLFetch() 186
SQLFreeHandle() 184
SQLGetConnectAttr 185
SQLGetDiagField 190
SQLGetDiagRec 190
SQLGetDiagRec() 190
SQLJ 4, 9
SQLj 239
sqlj 10
SQLJ translator 4
sqln 176
sqlname.data 176
SQLSetConnectAttr 185
SQLSetConnection 185
SQLSTATE 159
SQLState 228
SQLVAR 176
stack trace 229
statement 4, 148, 154
Statement object 209
static 148, 154
static cursor 91
static embedded SQL 4, 155
statistics 148
storage model 53
stored procedure 94, 219
StoredProc class 220
string value 60, 208
System.Data.ODBC 279
System.Data.OleDb 278

T

tables 259
text node 126
three tier model 50
tocAction() 104
toolbar 266
transaction 236
tree structure 55
Triggers View 260, 262
try-catch block 228
Type 1 8
Type 2 8, 200
Type 3 8
Type 4 8, 200

U

uncommitted read 236
unicode 91
unit of work 233
universal driver 200
universal JDBC driver 200
Unmanaged Provider 267
updatability 245
update_product() 154
updateTable() 201
UTF-8 224

V

VARCHAR 58
variable 151
views 259

W

Web services message 117
well-formed XML 55
WHENEVER clause 166
whitespace 60
wiki 97
workflow 98

X

XCS 116
XML 2, 191
XML Content Store 116
XML schema 51, 54
XML schema repository 54
XML value index 79

XMLContentStoreDB2 119
XmlDocument 289
XmlDocument() 289
XmlImplementation.CreateDocument() 289
XmlNamespaceManager 282
XmlNodeList 284
XMLPARSE 59
xmlpattern 79
XmlReader 282
XMLSERIALIZE 193
XPath 282
XQUERY 94
XQuery Visual Builder 24
XSR 54

Z

Zend_Controller 98
Zend_Db_Xml_Content 118



DB2 Express-C: The Developer Handbook for XML, PHP, C/C++, Java, and .NET

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



DB2 Express-C: The Developer Handbook for XML, PHP, C/C++, Java, and .NET



Learn DB2 application development with XML, PHP, C/C++, JAVA, and .NET

Understand DB2 supported programming environments

Practical application examples

DB2 Express Edition for Community (DB2 Express-C) is a no charge data server for use in development and deployment. DB2 Express-C supports a full range of APIs, drivers, and interfaces for application development including PHP, C/C++, and .NET. In addition, DB2 Express-C V9 contains advanced XML features. DB2 Express-C provides ISVs an ideal starting database server for Web, enterprise, and eBusiness applications.

This IBM Redbook provides fundamentals of DB2 application development with DB2 Express-C. It covers the DB2 Express-C installation and configuration for application development and skills and techniques for building DB2 applications with XML, PHP, C/C++, Java, and .NET.

Code examples are used to demonstrate how to develop a DB2 application in a different language. By following the examples provided, you will be able to learn DB2 application development with XML, PHP, C/C++, Java, and .NET in a short time.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks