

# Getting Started With

Hands-on  
exercises  
included

# DB2

## Application Development

Ideal for application developers and administrators



by:

**RAUL F. CHONG**

**XIQIANG JI**

**PRIYANKA JOSHI**

**VINEET MISHRA**

**MIN WEI YAO**

**DB2 ON CAMPUS** BOOK SERIES



GETTING STARTED WITH  
**DB2 application  
development**

A book for the community by the community

RAUL F. CHONG, XIQIANG JI, PRIYANKA JOSHI,  
VINEET MISHRA, MIN WEI YAO

---

**FIRST EDITION**

**First Edition (October 2010)**

**© Copyright IBM Corporation 2010. All rights reserved.**

IBM Canada  
8200 Warden Avenue  
Markham, ON  
L6G 1C7  
Canada

---

## Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan, Ltd.  
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711*

**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.

## 6 Getting started with DB2 application development

---

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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

### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.

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

## Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "[Copyright and trademark information](http://www.ibm.com/legal/copytrade.shtml)" at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

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

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

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

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





# Table of Contents

<b>Preface</b> .....	<b>15</b>
Who should read this book? .....	15
How is this book structured? .....	15
A book for the community .....	15
Conventions .....	16
What's next? .....	16
<b>About the authors</b> .....	<b>19</b>
<b>Contributors</b> .....	<b>20</b>
<b>Acknowledgements</b> .....	<b>21</b>
<b>Chapter 1 – Introduction to DB2 application development</b> .....	<b>23</b>
1.1 DB2 application development: The big picture .....	23
1.2 Server-side development .....	25
1.2.1 Stored procedures .....	25
1.2.2 User-defined functions .....	26
1.2.3 Triggers .....	26
1.3 Client-side development .....	27
1.3.1 Embedded SQL .....	27
1.3.2 Static SQL vs. Dynamic SQL .....	28
1.3.3 CLI and ODBC .....	30
1.3.4 JDBC, SQLJ and pureQuery .....	33
1.3.5 OLE DB .....	35
1.3.6 ADO.NET .....	36
1.3.7 PHP .....	37
1.3.8 Ruby on Rails .....	38
1.3.9 Perl .....	38
1.3.10 Python .....	38
1.4 XML and DB2 pureXML .....	39
1.5 Web services .....	40
1.6 Administrative APIs .....	41
1.7 Development tools .....	41
1.7.1 Visual Studio .....	42
1.7.2 Eclipse .....	42
1.7.3 Access and Excel .....	42
1.8 Development environments .....	43
1.8.1 DB2 Offerings on the Cloud .....	43
1.8.2 DB2 Express-C virtual appliance for VMWare .....	47
1.9 Sample programs .....	47
1.10 Exercises .....	47
1.11 Summary .....	48
1.12 Review questions .....	48
<b>Chapter 2 – DB2 pureXML</b> .....	<b>51</b>
2.1 Using XML with databases .....	52

2.2 XML databases .....	52
2.2.1 XML-enabled databases .....	52
2.2.2 Native XML databases .....	53
2.3 XML in DB2 .....	54
2.3.1 pureXML technology advantages .....	55
2.3.2 XPath basics .....	57
2.3.3 XQuery basics .....	60
2.3.4 Inserting XML documents .....	62
2.3.5 Querying XML data .....	65
2.3.6 Joins with SQL/XML .....	72
2.3.7 Joins with XQuery .....	73
2.3.8 Update and delete operations .....	74
2.3.9 XML indexing .....	76
2.4 Working with XML Schemas .....	77
2.4.1 Registering your XML Schemas .....	77
2.4.2 XML Schema validation .....	80
2.4.3 Other XML support .....	81
2.5 Exercises .....	82
2.6 Summary .....	83
2.7 Review questions .....	83
<b>Chapter 3 – Stored procedures, UDFs, triggers, and data Web services .....</b>	<b>85</b>
3.1 Stored procedures: The big picture .....	85
3.2 Working with IBM Data Studio .....	87
3.2.1 Creating a project .....	88
3.2.2 Creating a stored procedure .....	90
3.3 SQL PL stored procedures basics .....	94
3.3.1 Stored procedure structure .....	94
3.3.2 Optional stored procedure attributes .....	94
3.3.3 Parameters .....	95
3.3.4 Comments in an SQL PL stored procedure .....	96
3.3.5 Compound statements .....	96
3.3.6 Variable declaration .....	96
3.3.7 Assignment statements .....	97
3.3.8 Cursors .....	98
3.3.9 Flow control .....	98
3.3.10 Errors and condition handlers .....	99
3.3.11 Calling stored procedures .....	101
3.3.12 Dynamic SQL .....	102
3.4 Java Stored Procedures .....	103
3.5 User-defined functions: The big picture .....	105
3.5.1 Scalar functions .....	106
3.5.2 Table functions .....	107
3.6 Triggers: The big picture .....	107
3.6.1 Types of triggers .....	108

---

3.7 Data Web services .....	111
3.8 Exercises .....	121
3.9 Summary .....	123
3.10 Review questions .....	123
<b>Chapter 4 – Application development with Java .....</b>	<b>125</b>
4.1 Java - DB2 applications: The big picture .....	125
4.2 Setting up the environment .....	126
4.2.1 DB2 JDBC and SQLJ drivers .....	126
4.3 JDBC Programming .....	129
4.3.1 Connecting to a DB2 database .....	130
4.3.2 Executing SQL statements .....	132
4.3.3 Receiving results .....	142
4.3.4 Handling SQL errors and warnings .....	144
4.3.5 Closing the connection .....	146
4.3.6 Working with XML .....	146
4.4 SQLJ Programming .....	149
4.4.1 SQLJ Syntax .....	149
4.4.2 Connection contexts .....	150
4.4.3 Execution contexts .....	152
4.4.4 Iterators .....	153
4.4.5 Working with JDBC and SQLJ combined .....	155
4.4.6 Preparing an SQLJ program .....	156
4.5 pureQuery .....	159
4.6 Exercises .....	160
4.7 Summary .....	162
4.8 Review questions .....	162
<b>Chapter 5 – Application development with C/C++ .....</b>	<b>165</b>
5.1 C/C++ DB2 applications: The big picture .....	165
5.2 Setting up the environment .....	166
5.2.1 Supported compilers .....	166
5.2.2 Setting up the C/C++ environment .....	167
5.3 Developing a C/C++ application with embedded SQL .....	170
5.3.1 Source file extensions .....	170
5.3.2 SQL data types in C/C++ .....	171
5.3.3 Steps to develop an embedded SQL C/C++ application .....	172
5.3.4 Sample embedded SQL C/C++ application .....	174
5.3.5 Building embedded SQL C/C++ applications .....	185
5.5 Developing a C/C++ application with ODBC/CLI .....	191
5.5.1 Additional environment setup for CLI/ODBC applications .....	192
5.5.2 Handles .....	194
5.5.3 Steps to develop an ODBC/CLI application .....	195
5.5.4 Building ODBC/CLI applications .....	212
5.6 Working with XML in C/C++ applications with DB2 .....	214
5.7 Exercises .....	214

---

5.8 Summary.....	214
5.9 Review questions.....	215
<b>Chapter 6 – Application Development with .NET.....</b>	<b>217</b>
6.1 .NET with DB2 applications: The big picture .....	217
6.2 The ADO.NET data architecture.....	218
6.2.1 Data providers for ADO.NET .....	219
6.2.2 DataSet for ADO.NET.....	226
6.3 Setting up the environment.....	227
6.3.1 IBM Database Add-Ins for Visual Studio .....	228
6.3.2 Using Visual Studio with DB2 .....	231
6.4 Developing .NET - DB2 applications .....	235
6.4.1 Connecting to a DB2 database with the IBM Data Server Provider for .NET.....	238
6.4.2 Connecting to a DB2 database with the OLE DB .NET Data Provider .....	240
6.5 Data Manipulation using .NET .....	244
6.5.1 Building and Running the sample program .....	245
6.6 Exercises .....	246
6.7 Summary.....	246
6.8 Review questions.....	247
<b>Chapter 7 - Application development with Ruby on Rails.....</b>	<b>249</b>
7.1 Ruby on Rails applications with DB2: The big picture.....	249
7.2 Setting up the RoR environment.....	252
7.2.1 Installing Ruby .....	252
7.2.2 Installing Rails.....	255
7.2.3 Creating your first RoR application and starting the Web server .....	256
7.2.4 Working with a DB2 database: The ibm_db gem .....	258
7.3 Developing RoR applications.....	263
7.3.1 Developing a sample application: A book catalog.....	263
7.3.2 Customizing the layout .....	276
7.4 Exercises .....	281
7.5 Summary.....	282
7.6 Review questions.....	282
<b>Chapter 8 – Application development with PHP.....</b>	<b>285</b>
8.1 PHP - DB2 Applications: The big picture.....	285
8.2 Setting up the environment.....	286
8.2.1 Setting up the PHP environment manually.....	286
8.3 PHP - DB2 application development .....	289
8.3.1 PHP extensions to use with DB2 .....	289
8.3.2 PHP development with the ibm_db2 extension .....	289
8.3.3 PHP development with PDO_IBM/PDO_ODBC.....	300
8.4 Optimizing DB2 usage with PHP .....	318
8.4.1 Design considerations for increasing the PHP-DB2 performance .....	318
8.5 Exercises .....	319
8.6 Summary.....	319
8.7 Review questions.....	319

---

<b>Chapter 9 – Application development with Perl .....</b>	<b>321</b>
9.1 Perl - DB2 applications: The big picture .....	321
9.2 Setting up the environment.....	322
9.2.1 Perl adapters and drivers.....	324
9.3 Developing Perl DB2 applications .....	325
9.3.1 Connecting to a DB2 database.....	325
9.3.2 Retrieving data.....	326
9.3.3 Inserting, updating, and deleting data .....	328
9.3.4 Executing a SQL statement with parameter markers.....	330
9.3.5 Calling a stored procedure.....	331
9.4 Exercises .....	334
9.5 Summary.....	336
9.6 Review questions.....	336
<b>Chapter 10 –Application development with Python .....</b>	<b>337</b>
10.1 Python - DB2 applications: The big picture .....	337
10.1.1 IBM defined API and <code>ibm_db</code> driver.....	338
10.1.2 Python Database API <i>and</i> <code>ibm_db_dbi driver</code> .....	338
10.1.3 SQLAlchemy <i>and</i> <code>ibm_db_sa adapter</code> .....	339
10.1.4 Django framework <i>and</i> <code>ibm_db_django adapter</code> .....	339
10.2 Setting up the environment.....	339
10.2.1 Python adapters & drivers .....	340
10.3 Developing Python DB2 applications.....	347
10.3.1 Connecting to a DB2 database.....	347
10.3.2 Retrieving data.....	348
10.3.3 Inserting, updating and deleting data .....	351
10.3.4 Execute a SQL statement with parameter markers.....	352
10.3.5 Call a stored procedure .....	355
10.4 Exercises .....	358
10.5 Summary.....	358
10.6 Review questions.....	358
<b>Appendix A – Solutions to the review questions .....</b>	<b>361</b>
<b>Appendix B – Troubleshooting.....</b>	<b>369</b>
B.1 Finding more information about error codes .....	370
B.2 SQLCODE and SQLSTATE .....	370
B.3 DB2 Administration Notification Log.....	371
B.4 <code>db2diag.log</code> .....	371
B.5 CLI traces .....	372
B.6 DB2 Defects and Fixes.....	372
<b>References.....</b>	<b>373</b>
<b>Resources.....</b>	<b>373</b>
Web sites .....	373
Books .....	375
Contact emails .....	375



# Preface

Keeping your skills current in today's world is becoming increasingly challenging. There are too many new technologies being developed, and little time to learn them all. The DB2® on Campus Book Series has been developed to minimize the time and effort required to learn many of these new technologies.

## Who should read this book?

This book is intended for anyone who works with or intends to develop database applications such as application developers, consultants, software architects, instructors, and students. It is a good reference as well for database administrators (DBAs) and product managers.

## How is this book structured?

This book is closely related to the eBook *Getting Started with DB2 Express-C*; it expands the application development chapters covered in that book. In fact, Chapter 1 and 2 are taken directly from the application development chapters in that book as they provide a good overview of DB2 application development. Chapter 3 discusses server-side programming such as stored procedures, and functions. In this chapter IBM® Data Studio software is used extensively, therefore this eBook is also closely related to eBook *Getting Started with IBM Data Studio for DB2*. Starting with Chapter 4 the book describes in detail client-side programming for different programming languages such as Java™, C/C++, .NET, Ruby on Rail, PHP, Perl, and Python.

Exercises are provided with most chapters; any input files required are provided in the zip file `Exercise_Files_DB2_Application_Development.zip` accompanying this book.

## A book for the community

This book was created by the community; a community consisting of university professors, students, and professionals (including IBM employees). The online version of this book is released to the community at no-charge. Numerous members of the community from around the world have participated in developing this book, which will also be translated to several languages by the community. If you would like to provide feedback, contribute new material, improve existing material, or help with translating this book to another language, please send an email of your planned contribution to [db2univ@ca.ibm.com](mailto:db2univ@ca.ibm.com) with the subject "Getting Started with DB2 Application Development book feedback."

## Conventions

Many examples of commands, SQL statements, and code are included throughout the book. Specific keywords are written in uppercase bold. For example: A **NULL** value represents an unknown state. Commands are shown in lowercase bold. For example: The **dir** command lists all files and subdirectories on Windows®. SQL statements are shown in upper case bold. For example: Use the **SELECT** statement to retrieve information from a table.

Object names used in our examples are shown in bold italics. For example: The ***flights*** table has five columns.

Italics are also used for variable names in the syntax of a command or statement. If the variable name has more than one word, it is joined with an underscore. For example:  
**CREATE TABLE *table\_name***

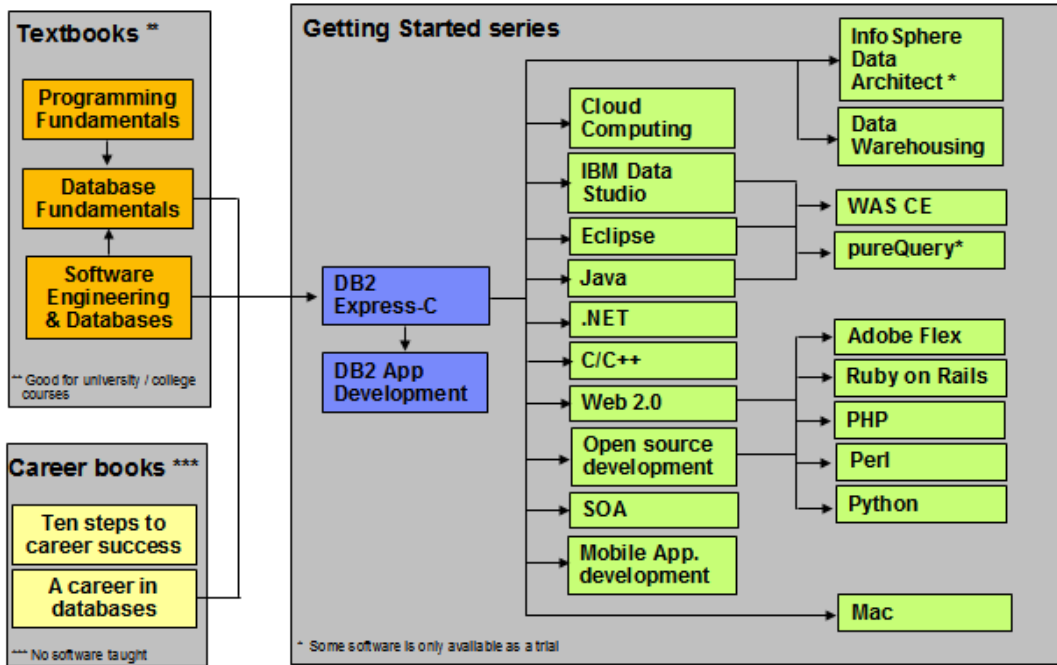
## What's next?

We recommend you to review the following books in this book series for more details about related topics:

- *Getting started with DB2 Express-C*
- *Getting started with IBM Data Studio for DB2*
- *Getting started with Java*
- *Getting started with C/C++*
- *Getting started with .NET*
- *Getting started with Ruby on Rails*
- *Getting started with PHP*
- *Getting started with Perl*
- *Getting started with Python*
- *Getting started with Open source development*
- *Getting started with Eclipse*

The following figure shows all the different eBooks in the DB2 on Campus book series available for free at [ibm.com/db2/books](http://ibm.com/db2/books)





The DB2 on Campus book series



## About the authors

**Raul F. Chong** is the DB2 on Campus program manager and a DB2 technical evangelist based at the IBM Toronto Laboratory. His main responsibility is to grow the DB2 community around the world. Raul joined IBM in 1997 and has held numerous positions in the company. As a DB2 consultant, Raul helped IBM business partners with migrations from other relational database management systems to DB2, as well as with database performance and application design issues. As a DB2 technical support specialist, Raul helped resolve DB2 problems on the OS/390®, z/OS®, Linux®, UNIX® and Windows® platforms. Raul has taught many DB2 workshops, has published numerous articles, and has contributed to the DB2 Certification exam tutorials. Raul has summarized many of his DB2 experiences through the years in his book *Understanding DB2 - Learning Visually with Examples 2nd Edition (ISBN-10: 0131580183)* for which he is the lead author. He has also co-authored the book *DB2 SQL PL Essential Guide for DB2 UDB on Linux, UNIX, Windows, i5/OS, and z/OS (ISBN 0131477005)*, and is the project lead and co-author of many of the books in the DB2 on Campus book series.

**Xiqiang Ji** is a DB2 Advanced Support Engineer in IBM AP DB2 Level 2 support team in Sydney. His main responsibility is to provide technical support for IBM Asia Pacific and worldwide customers for solving various DB2 problems. During the past 5 years, He has helped many DB2 customers across various industries solve many critical technical issues. Before this, He had worked for 5 years as a technical consultant in IBM Software Group supporting IBM Business Partners and Independent Software Vendors in developing DB2 applications and DB2 Business Intelligence solutions.

**Priyanka Joshi** is a software engineer with IBM India software labs working as a DB2 advanced technical support specialist. Her primary responsibility is to provide advanced technical support on DB2 Linux®, UNIX® and Windows (LUW) platforms to IBM worldwide customers. Priyanka joined IBM in 2006 and has since worked for numerous pre-sales and post-sales support engagements for DB2 LUW. She specializes in DB2 - Common Client Technologies and is a certified DB2 professional. Priyanka has been identified as the Knowledge Champion for Asia-Pacific division as part of the Knowledge Centered support initiative in IBM and also is a part of the prestigious Technical Leaders group in IBM, responsible for providing smart solutions to IBM customers in collaboration with other IBM product teams.

**Vineet Mishra** is a software engineer with the DB2 LUW team at the India Software Lab. Vineet Joined IBM in 2007 and specializes in C and C++. His areas of interest are High Availability and Disaster Recovery (HADR), stored procedures & UDFs, Embedded SQL and Operating System Kernel. Vineet is a member of IBM Academic Initiative and IBM University Relationship and actively works towards spreading DB2 (LUW) knowledge in colleges. He frequently responds to queries in the DB2 forum.

**Min Wei Yao** is an application developer focusing on Business Intelligence. Min Wei joined IBM in 2008 and has been working in the IBM Global Business Services area since then. Besides working as a developer, Min Wei also likes to experiment with Linux, and DB2. Min

Wei is an IBM certified DB2 application developer and administrator for Linux, UNIX and Windows.

## Contributors

The following people edited, reviewed, provided content, and contributed significantly to this book.

<b>Contributor</b>	<b>Company / University</b>	<b>Position / Occupation</b>	<b>Contribution</b>
Antonio Cangiano	IBM Toronto Lab	Software Engineer and Technical Evangelist	Partial technical review
Praveen Devarao	IBM India Software Lab	Software Engineer, IBM OpenSource Technologies for IBM Data Servers	Partial technical review
Vinay B. Ganapavarapu	University of New Mexico	Student	Partial technical review
Upal Hossain	IBM Toronto Lab	Software Developer, DB2 Information Development Infrastructure	Partial technical review
Leon Katsnelson	IBM Toronto Lab	Program Director, IBM Data Servers	Technical review
Anil Mahadev	IDUG India	IDUG India chairman, database consultant	Partial technical review
Leons Petrazickis	IBM Toronto Lab	Software Developer and Technical Evangelist	Partial technical review
Rahul Priyadarshi	IBM India Software Lab	System Software Engineer, IBM open source Technologies for IBM Data Servers	Partial technical review

## **Acknowledgements**

We greatly thank the following individuals for their assistance in developing materials referenced in this book:

- Natasha Tolub who designed the cover of this book.
- Susan Visser who assisted with publishing this book.



# 1

## Chapter 1 – Introduction to DB2 application development

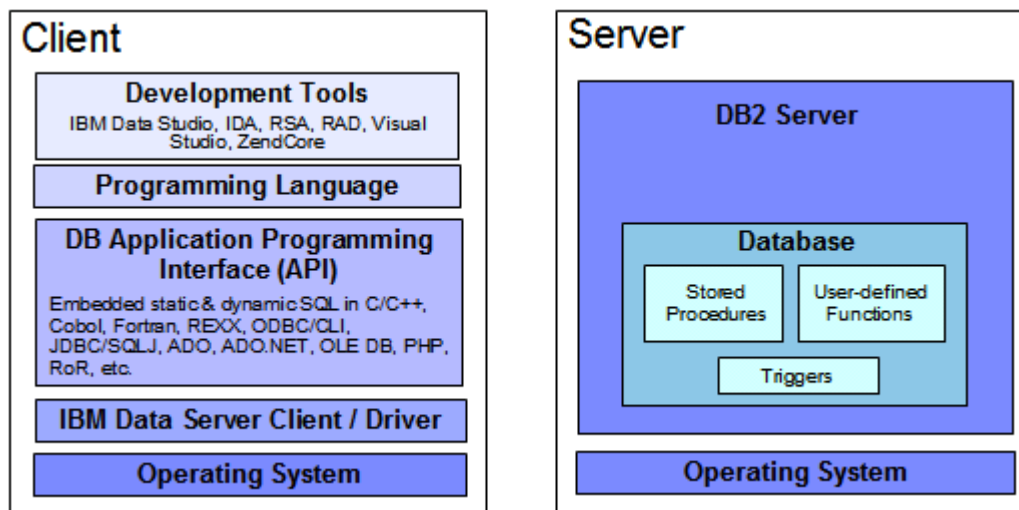
IBM® DB2® is powerful data server software for managing both relational and XML data. It offers flexibility not only to database administrators, but also to database developers. No matter which language you use to develop your programs, DB2 software ("DB2") provides the drivers, adapters, and extensions you need to work with databases as part of your application. Moreover with DB2 Express-C, you can develop your applications at no cost, with no database size limits, and with the same level of programming language support as the other versions of DB2. Develop once using DB2 Express-C, and you can run on any DB2 edition without any modification required to your application.

In this chapter you will learn about:

- Server-side programming using stored procedures, and user-defined functions
- Client-side programming using different programming languages

### 1.1 DB2 application development: The big picture

DB2 offers database developers the flexibility to take advantage of server-side development features such as stored procedures and user-defined functions, while, application developers can develop client applications using the programming language of their choice. This flexibility is illustrated in *Figure 1.1*.



**Figure 1.1 - DB2 software is for everyone: Database and application developers**

In *Figure 1.1* the left side represents a client machine where an application programmer develops and runs his program. In this client machine, in addition to the operating system, an IBM Data Server Client may be installed depending on the type of application being developed. An IBM Data Server client includes the required connection drivers such as the JDBC drivers and the ODBC/CLI drivers. These drivers can also be downloaded independently by visiting the IBM DB2 Express-C Web site at <http://ibm.com/db2/express>

Using programming tools such as IBM Data Studio, InfoSphere™ Data Architect (IDA), Rational® Software Architect (RSA), Rational Application Developer (RAD), and so on, you can develop your application in your desired programming language. The API libraries supporting these languages are also included with the IBM Data Server Client, so that when you connect to a DB2 Server, all the program instructions are translated appropriately using these APIs into the SQL or XQuery statements understood by DB2. *Table 1.1* provides a short description of the tools mentioned earlier.

Tool name	Description
IBM Data Studio	IBM Data Studio is a free Eclipse-based tool that allows users to manage their data servers and develop stored procedures, functions and Data Web services. For more details, refer to the ebook <i>Getting started with IBM Data Studio for DB2</i> .
InfoSphere Data Architect (IDA)	IDA is a modeling tool for your data. It helps you build your database logical design and physical design. For more details, refer to



	the ebook <i>Getting started with InfoSphere Data Architect</i> .
Rational Software Architect (RSA)	RSA is an Eclipse-based tool for software engineering to help you develop UML diagrams
Rational Application Developer (RAD)	RAD is an Eclipse-based rapid application development tool for software developers
Visual Studio	Microsoft® Visual Studio is an IDE that allows you to develop applications in the Windows® platform using Microsoft's technology.
Aptana Studio	This is a free IDE for developing PHP applications.

**Table 1.1 - Tools that can help you develop applications with DB2 software**

On the right side of *Figure 1.1* a DB2 server is illustrated containing one database. Within this database there are stored procedures, user-defined functions and triggers. We describe all of these objects in more detail in the next sections.

It is noteworthy to mention that IBM offers DB2 on the Amazon cloud, as well as on the IBM Development and Test Cloud. If you or your company does not have the budget to acquire a server for your development or production needs, the Cloud is a perfect alternative as it allows you to "rent" compute capacity per minute. DB2 on the Cloud offerings are discussed in more detail in a later section.

## 1.2 Server-side development

Server-side development in DB2 software implies that application objects are developed and stored on the DB2 database. The following application objects will be discussed briefly in this section:

- Stored Procedures
- User-defined Functions (UDFs)
- Triggers

### 1.2.1 Stored procedures

A **stored procedure** is a database application object that can encapsulate SQL statements and business logic. Keeping part of the application logic in the database provides performance improvements as the amount of network traffic between the application and the database is reduced. In addition, stored procedures provide a centralized location to

store your code, so other applications can reuse the same stored procedures. To call a stored procedure, use the **CALL** statement. In DB2 you can develop stored procedures in several languages including SQL PL, PL/SQL, Java, C/C++, CLR, OLE, and COBOL. A simple example of how to create and call a SQL PL stored procedure from the DB2 Command Window or Linux® shell is shown below:

```
db2 create procedure P1 begin end
db2 call P1
```

In the example, procedure P1 is an empty stored procedure which is not doing anything. The example illustrates how easy you can create a stored procedure. To develop stored procedures with more complex logic, we recommend you use IBM Data Studio which includes a debugger.

### 1.2.2 User-defined functions

A **user-defined function (UDF)** is a database application object that allows users to extend the SQL language with their own logic. A function always returns a value or values normally as a result of the business logic included in the function. To invoke a function, use it within a SQL statement, or with the **VALUES** function. In DB2 you can develop UDFs in several languages including SQL PL, PL/SQL, Java, C/C++, OLE DB, CLR.

This simple example shows how to create and call a SQL PL UDF from the DB2 Command Window or Linux shell:

```
db2 create function F1() returns integer begin return 1000; end
db2 values F1
```

In the example, function F1 is a function returning an integer value of 1000. The **VALUES** statement can be used to invoke the function. Like in the case of stored procedures, we recommend you create functions using IBM Data Studio.

### 1.2.3 Triggers

A **trigger** is an object that automatically performs an operation on a table or view. A triggering action on the object where the trigger is defined causes the trigger to be fired. A trigger is normally not considered an application object; therefore, database developers normally don't code triggers, but database administrators do. Because some coding is required, we have included triggers in this section. Below is an example of a trigger:

```
create trigger myvalidate no cascade before insert on T1
referencing NEW as N
for each row
begin atomic
set (N.myxmlcol) = XMLVALIDATE(N.myxmlcol
according to xmlschema id myxmlschema);
```

end

In this example, the trigger is fired before an INSERT operation on table T1. The trigger will insert the value (which is an XML document), but will invoke the XMLVALIDATE function to validate this XML document with a given schema. *Chapter 15, DB2 pureXML* talks more about XML and XML schemas.

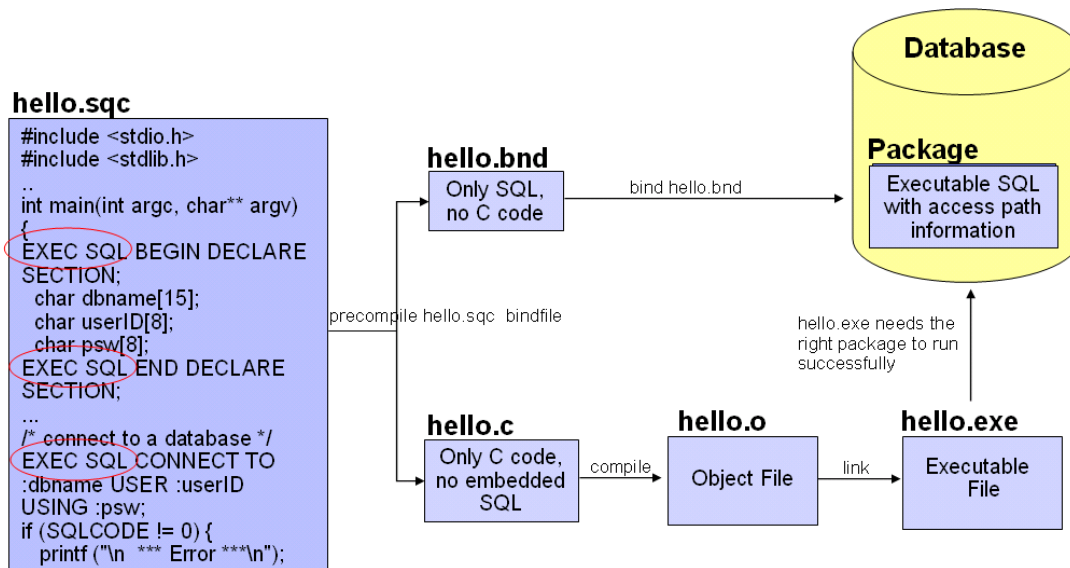
### 1.3 Client-side development

As the name suggests, in client-side development, the application developers code their programs on a client and then connect and access the DB2 database using the application program interfaces (APIs) that are provided with DB2. In this section we discuss:

- Embedded SQL
- Static SQL vs Dynamic SQL
- CLI and ODBC
- JDBC, SQLJ and pureQuery
- OLE DB
- ADO.NET
- PHP
- Ruby on Rails
- Perl
- Python

#### 1.3.1 Embedded SQL

***Embedded SQL applications*** are applications where SQL is embedded into a host language such as C, C++, or COBOL. The embedded SQL application can include static or dynamic SQL as described in the next section. *Figure 1.2* shows how an embedded SQL application is built.



**Figure 1.2 - Building embedded SQL applications**

In the figure, the C program `hello.sqc` contains embedded SQL. The embedded SQL API for the C language uses `EXEC SQL` (highlighted in *Figure 1.2*) to allow a precompilation process to distinguish between the embedded SQL statements and the actual C code. You may also note in the `hello.sqc` listing that some variables are prefixed with a colon, as in `:dbname`, `:userID`, and `:psw`. These are called host variables. Host variables are variables from the host language that are referenced in the embedded SQL statements.

Issuing the `precompile` command (also known as the `prep` command) with the `bindfile` option generates two files, the `hello.bnd` bind file containing only SQL statements and the `hello.c` file containing only C code. The bind file will be compiled using the `bind` command to obtain a **package** that is stored in the database. A package includes the compiled/executable SQL and the access path DB2 will follow to retrieve the data. To issue the `bind` command, a connection to the database must exist. At the bottom of the figure, the `hello.c` file is compiled and linked like any regular C program. The resulting executable file `hello.exe` has to match the package stored in the database to successfully execute.

### 1.3.2 Static SQL vs. Dynamic SQL

**Static SQL** statements are the ones where the SQL structure is fully known at precompile time. For example:

```
SELECT lastname, salary FROM employee
```

In this example, the names for the columns (**lastname**, **salary**) and table (**employee**) referenced in a statement are fully known at precompile time. The following example is also a static SQL statement:

```
SELECT lastname, salary FROM employee WHERE firstnme = :fname
```

In this second example, a host variable **:fname** is used as part of an embedded SQL statement. Though the value of the host variable is unknown until runtime, its data type is known from the program, and all the other objects (column names, table names) are fully known ahead of time. DB2 software uses estimates for these host variables to calculate the access plan ahead of time; therefore, this case is still considered static SQL.

You precompile, bind, and compile statically executed SQL statements before you run your application. Static SQL is best used on databases whose statistics do not change a great deal. Now let's take a look at one more example:

```
SELECT ?, ? FROM ?
```

In this example, the names for the columns and table referenced by the statement are not known until runtime. Therefore the access plan is calculated only at runtime and using the statistics available at the time. These types of statements are considered **Dynamic SQL** statements.

Some programming APIs, like JDBC and ODBC, always use dynamic SQL regardless of whether the SQL statement includes known objects or not. For example, the statement **SELECT lastname, salary FROM employee** has all the columns and table names known ahead of time, but through JDBC or ODBC, you do not precompile the statements. All the access plans for the statements are calculated at runtime.

In general, two statements are used to treat a SQL statement as dynamic:

- **PREPARE**: This statement prepares or compiles the SQL statement calculating the access plan to use to retrieve the data
- **EXECUTE**: This statement executes the SQL

Alternatively you can execute a **PREPARE** and **EXECUTE** in one single statement: **EXECUTE IMMEDIATELY**

*Listing 1.1* shows an example on an embedded C dynamic SQL statement that is prepared and executed.

```
strcpy(hVStmtDyn, "SELECT name FROM emp WHERE dept = ?");
PREPARE StmtDyn FROM :hVStmtDyn;
EXECUTE StmtDyn USING 1;
EXECUTE StmtDyn USING 2;
```

#### **Listing 1.1 - An embedded C dynamic SQL statement using PREPARE and EXECUTE**

*Listing 1.2* shows the same example as *Listing 1.1*, but using the **EXECUTE IMMEDIATELY** statement

```
EXECUTE IMMEDIATELY SELECT name from EMP where dept = 1
EXECUTE IMMEDIATELY SELECT name from EMP where dept = 2
```

**Listing 1.2 - An embedded C dynamic SQL statement using EXECUTE IMMEDIATELY**

In many dynamic programming languages such as PHP or Ruby on Rails, where SQL is run dynamically, programmers tend to write the same SQL statements with different field values as follows:

```
SELECT lastname, salary FROM employee where firstnme = 'Raul'
SELECT lastname, salary FROM employee where firstnme = 'Jin'
...
```

In this example, the statements are identical except for the value of the column **firstnme**. DB2 considers these two dynamic SQL statements as different ones, and therefore at runtime, it prepares and then executes each statement independently. The overhead of preparing the same statement several times can cause performance degradation, therefore prior to DB2 9.7, the recommendation was to code statements as follows:

```
SELECT lastname, salary FROM employee where firstnme = ?
```

The question mark (?) in the statement is known as a **parameter marker**. Using parameter markers, the program could prepare the statement only once, and then issue **EXECUTE** statements providing the different values for the parameter marker.



In DB2 9.7, DB2 introduced a technology called **statement concentrator** where all the statements that are the same except for the field values are automatically lumped together into one single statement with parameter markers, and then **EXECUTE** statements are performed with the different values. The statement concentrator does have the intelligence to determine when not to lump some statements together; for example, when you purposely add some clauses to influence the DB2 optimizer.

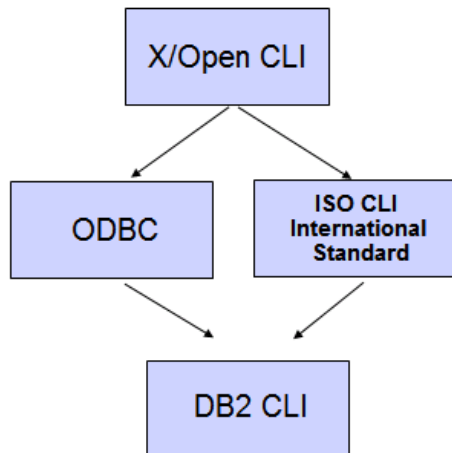
With respect to performance, static SQL will normally perform better than dynamic SQL since the access plan in static SQL is performed at precompile time and not at runtime. However, for environments where there is a lot of activity such as **INSERTs** and **DELETEs**, the statistics calculated at precompile time may not be up-to-date, and therefore, the access plan of the static SQL may not be optimal. In this case, dynamic SQL may be a better choice, assuming a **RUNSTATS** command is frequently executed to collect current statistics.

**Note:**  
Many users think embedded SQL is only static. In reality, it can be both, static or dynamic.

### 1.3.3 CLI and ODBC

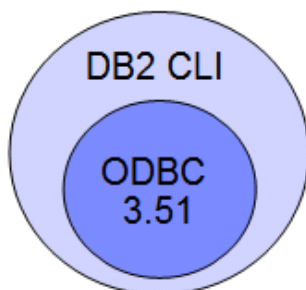
**Call Level Interface (CLI)** was originally developed by the X/Open Company and the SQL Access Group. It was a specification for a callable SQL interface with the purpose of

developing portable C/C++ applications regardless of the RDBMS vendor. Based on a preliminary draft of X/Open Call Level Interface, Microsoft developed **Open Database Connectivity (ODBC)**, and later on, the ISO CLI International Standard accepted most of the X/Open Call Level Interface specification. DB2 CLI is based on both: ODBC and the International Standard for SQL/CLI as shown in *Figure 1.3*.



**Figure 1.3 - DB2 CLI is based on ODBC and ISO CLI International standard**

DB2 CLI conforms to ODBC 3.51 and can be used as the ODBC Driver when loaded by an ODBC Driver Manager. Figure 1.4 can help you picture DB2 CLI support for ODBC.



**Figure 1.4 - DB2 CLI conforms to ODBC 3.51**

CLI/ODBC has the following characteristics:

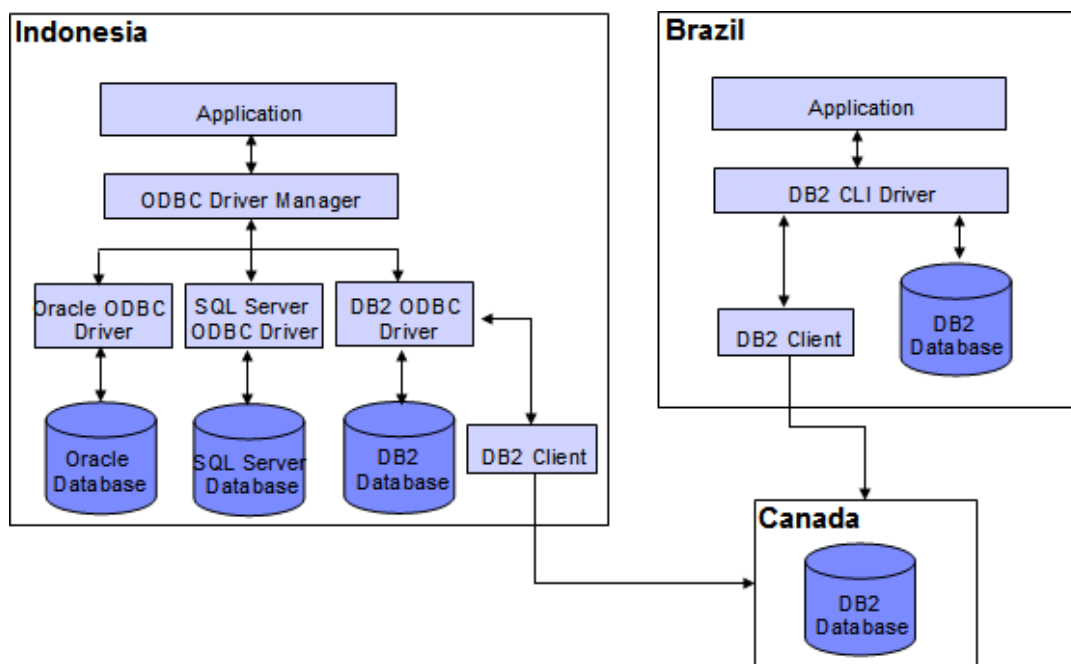
- The code is easily portable between several RDBMS vendors
- Unlike embedded SQL, there is no need for a precompiler or host variables
- It runs dynamic SQL
- It is very popular

To **run** a CLI/ODBC application all you need is the DB2 CLI driver. This driver is installed from either of the following clients and drivers which can be downloaded and used for free from [www.ibm.com/db2/express](http://www.ibm.com/db2/express):

- IBM Data Server Client
- IBM Data Server Runtime Client
- IBM Data Server Driver for ODBC and CLI

To **develop** a CLI/ODBC application you need the DB2 CLI driver and also the appropriate libraries. These can be found only on the IBM Data Server Client.

Let's take a look at the following example so you understand better how you can set up the DB2 CLI driver for your applications. *Figure 1.5* depicts three different machines, one in Indonesia, the other one in Brazil, and the other one in Canada.



**Figure 1.5 - DB2 CLI/ODBC sample scenario**

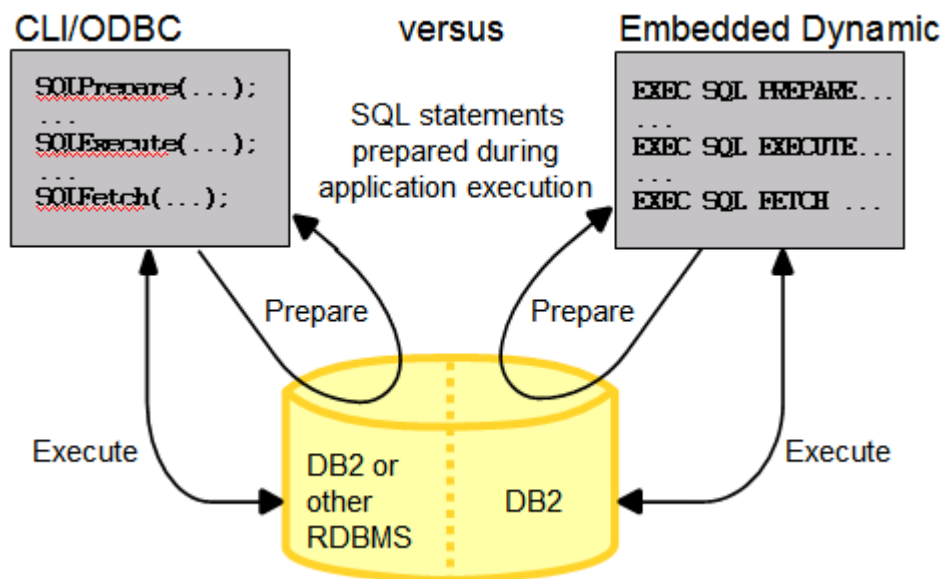
The figure shows two cases:

On the left let's say the machine in Indonesia is running an *ODBC application* which could work with any RDBMS such as Oracle®, Microsoft® SQL Server® or DB2 database server. An ODBC Driver Manager will load the appropriate ODBC driver depending on the database that is being accessed. In the case where the application accesses a DB2 database in Canada, the connection needs to go through a DB2 Client which has the components to connect remotely.



On the right side, let's say a *CLI application* is running in a machine in Brazil. It's a CLI application because it may be using some specific functions not available in ODBC, and also because the application will only work for a DB2 database. The CLI application will go through the DB2 CLI Driver. The application can connect to the local DB2 database in Brazil. When it needs to connect to the remote database in Canada, it will go through a DB2 client.

One last point to be made in this section is a comparison between a CLI/ODBC application and an embedded SQL C dynamic application. *Figure 1.6* illustrates this comparison.



**Figure 1.6 - CLI/ODBC application versus Embedded SQL C dynamic application**

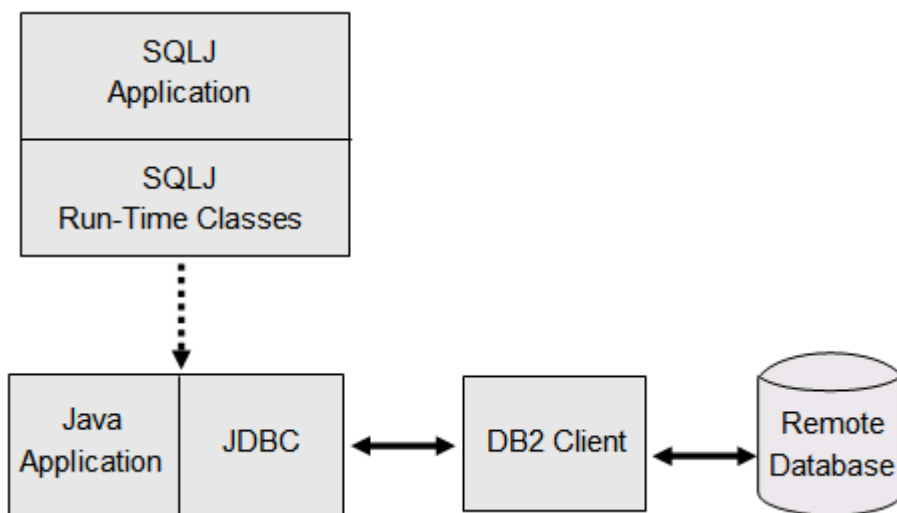
As shown in *Figure 1.6*, the only difference between CLI/ODBC vs. Embedded SQL C dynamic SQL is that for CLI/ODBC your code is portable and can access other RDBMS simply by changing the connection string, while in the embedded SQL C dynamic version, you may be coding specific elements for DB2. Of course the other difference is the way the different functions for **PREPARE**, and **EXECUTE** are invoked.

### 1.3.4 JDBC, SQLJ and pureQuery

**Java Database Connectivity (JDBC)** is a Java programming API that standardizes the means to work and access databases. In JDBC the code is easily portable between several RDBMS vendors. The only changes required to the code are normally which JDBC driver to load and the connection string. JDBC uses only dynamic SQL and it is very popular.

**SQLJ** is the standard for embedding SQL in Java programs. It is mainly used with static SQL, though it can inter-operate with JDBC as shown in *Figure 1.7*. Though it is normally more compact than JDBC programs and provides better performance, it has not been

widely accepted. SQLJ programs must be run through a preprocessor (the SQLJ translator) before they can be compiled.



**Figure 1.7 - Relationship between SQLJ and JDBC applications**

In *Figure 1.7*, a DB2 client may or may not be required depending on the type of JDBC driver used as discussed later on this section.

**pureQuery** is an IBM Eclipse-based plug-in to manage relational data as objects. Available since 2007, pureQuery can automatically generate the code to establish an object-relational mapping (ORM) between your object oriented code and the relational database objects. You start by creating a Java project with Optim™ Development Studio (ODS), connect to a DB2 database, and then have ODS discover all the database objects. Through the ODS GUI you can pick a table and then choose to generate the pureQuery code which would transform any of the underlying relational table entities into a Java object. Code is generated to create the relevant SQL statements and parent Java objects that encapsulate those statements. The generated Java objects and the contained SQL statements can be further customized. With pureQuery, you can decide at runtime whether you want to run your SQL in static or dynamic mode. pureQuery supports both Java and .NET.

#### 1.3.4.1 JDBC and SQLJ drivers

Though there are several types of JDBC drivers such as type 1, 2, 3 and 4; type 1 and 3 are not commonly used, and DB2's support of these types has been deprecated. For type 2, there are two drivers as we will describe shortly, but one of them is also deprecated.

Type 2 and type 4 are supported with DB2 software, as shown in *Table 1.2*. Type 2 drivers need to have a DB2 client installed, as the driver uses it to establish communication to the database. Type 4 is a pure Java client, so there is no need for a DB2 client, but the driver must be installed on the machine where the JDBC application is running.

Driver Type	Driver Name	Packaged as	Supports	Minimum level of SDK for Java required
Type 2	DB2 JDBC Type 2 Driver for Linux, UNIX® and Windows ( <b>Deprecated*</b> )	db2java.zip	JDBC 1.2 and JDBC 2.0	1.4.2
Type 2 and Type 4	IBM Data Server Driver for JDBC and SQLJ	db2jcc.jar and sqlj.zip	JDBC 3.0 compliant	1.4.2
		db2jcc4.jar and sqlj4.zip	JDBC 4.0 and earlier	6

**Table 1.2 - DB2 JDBC and SQLJ drivers**

\* Deprecated means it is still supported, but no longer enhanced

As mentioned earlier and shown also in *Table 1.2*, Type 2 is provided with two different drivers; however the DB2 JDBC Type 2 Driver for Linux, UNIX and Windows, with filename db2java.zip is deprecated.

When you install a DB2 server, a DB2 client or the IBM Data Server Driver for JDBC and SQLJ, the db2jcc.jar and sqlj.zip files compliant with JDBC 3.0 are automatically added to your classpath.

### 1.3.5 OLE DB

**Object Linking and Embedding, Database (OLE DB)** is a set of interfaces that provides access to data stored in diverse sources. It was designed as a replacement to ODBC, but extended to support a wider variety of sources, including non-relational databases, such as object oriented databases and spreadsheets. OLE DB is implemented using the Component Object Model (COM) technology.

OLE DB consumers can access a DB2 database with the IBM OLE DB Provider for DB2. This provider has the following characteristics:

- Provider name: IBMDADB2
- Supports level 0 of the OLE DB provider specification, including some additional level 1 interfaces
- Complies with Version 2.7 or later of the Microsoft OLE DB specification
- An IBM Data Server Client with the Microsoft Data Access Components (MDAC) must be installed

- 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 non-Microsoft SQL server data sources using the ODBC driver but does not guarantee full functionality of the OLE DB driver.

### 1.3.6 ADO.NET

The **.NET Framework** is the Microsoft replacement for Component Object Model (COM) technology. Using the .NET Framework, you can code .NET applications in over forty different programming languages; the most popular ones being C# and Visual Basic .NET.

The .NET Framework class library provides the building blocks with which you build .NET applications. This class library is language agnostic and provides interfaces to operating system and application services. Your .NET application (regardless of language) compiles into Intermediate Language (IL), a type of bytecode.

The Common Language Runtime (CLR) is the heart of the .NET Framework, compiling the IL code on the fly, and then running it. In running the compiled IL code, the CLR activates objects, verifies their security clearance, allocates their memory, executes them, and cleans up their memory once execution is finished.

As an analogy to how Java works, in Java, a program can run in multiple platforms with minimal or no modification: one language, but multiple platforms. In .NET, a program written in any of the forty supported languages can run in one platform, Windows, with minimal or no modification: multiple languages, but one platform.

**ADO.NET** is how data access support is provided in the .NET Framework. ADO.NET supports both connected and disconnected access. The key component of disconnected data access in ADO.NET is the `DataSet` class, instances of which act as a database cache that resides in your application's memory.

For both connected and disconnected access, your applications use databases through what is known as a **data provider**. Various database products include their own .NET data providers, including DB2 for Windows.

A .NET data provider features implementations of the following basic classes:

- Connection: establishes and manages a database connection.
- Command: executes an SQL statement against a database.
- DataReader: reads and returns result set data from a database.
- DataAdapter: links a DataSet instance to a database. Through a DataAdapter instance, the DataSet can read and write database table data.

Three data providers that can work with DB2 software are shown in *Table 1.3*

Data Provider	Characteristics
---------------	-----------------

<p>ODBC .NET Data provider (not recommended)</p>	<ul style="list-style-type: none"> <li>▪ Makes ODBC calls to a DB2 data source using the DB2 CLI driver.</li> <li>▪ It has same keyword support and restrictions as that of DB2 CLI driver</li> <li>▪ Can be used with .NET Framework Version 1.1, 2.0, or 3.0.</li> </ul>
<p>OLE DB .NET Data provider (not recommended)</p>	<ul style="list-style-type: none"> <li>▪ Uses IBM DB2 OLE DB Driver (IBMDADB2).</li> <li>▪ It has same keyword support and restrictions as that of DB2 OLE DB driver</li> <li>▪ Can be used only with .NET Framework Version 1.1, 2.0, or 3.0.</li> </ul>
<p>DB2 .NET Data provider (recommended)</p>	<ul style="list-style-type: none"> <li>▪ Extends DB2 support for the ADO.NET interface.</li> <li>▪ The DB2 managed provider implements the same set of standard ADO.NET classes and methods</li> <li>▪ It is defined under IBM.DATA.DB2 namespace.</li> <li>▪ Can be obtained by downloading any of: <ul style="list-style-type: none"> <li>- Data Server Driver for ODBC, CLI, and .NET</li> <li>- IBM Data Server Runtime Client</li> <li>- DB2 Data Server</li> </ul> </li> </ul>

**Table 1.3 - ADO.NET data providers**

### 1.3.7 PHP

**PHP Hypertext Preprocessor (PHP)** is an open source, platform independent scripting language designed for Web application development. It can be embedded within HTML, and generally runs on a Web server which takes the PHP code and creates Web pages as output.

PHP is a modular language. You can use extensions to customize the available functionality. Some of the most popular PHP extensions are those used to access databases. IBM supports access to DB2 databases 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\_ibm**: The pdo\_ibm 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.

The PHP extensions and drivers are available for free from the PECL repository at <http://pecl.php.net/> Windows builds of the extensions and drivers are available at <http://sourceforge.net/projects/db2mc/files/>

You will need the IBM Data Server Driver for ODBC and CLI to install the PHP extensions on Linux and UNIX. Both, ibm\_db2 and pdo\_ibm are based on the IBM DB2 CLI Layer.

### 1.3.8 Ruby on Rails

Ruby is an open source object oriented language. Rails is a Web framework created using Ruby. Ruby on Rails (RoR) is an ideal means to develop database backed web-based applications. This hot new technology is based on the Model, View, Controller (MVC) architecture and follows the principles of agile software development.

Rails requires no special file formats or integrated development environments (IDEs); you can get started with a text editor. However, various IDEs are available with Rails support, such as RadRails, which is a Rails environment for Eclipse. For more information about RadRails, visit <http://www.radrails.org/>.

DB2 supports Ruby 1.8.5 and later and Ruby on Rails 1.2.1 and later. The IBM\_DB gem includes the IBM\_DB Ruby driver and Rails adapter which allows you to work with DB2 and is based on the CLI layer. This gem must be installed along with an IBM Data Server Client. To install the IBM\_DB driver and adapter you can use Ruby gem or as a Rails plug-in.

### 1.3.9 Perl

Perl is a popular interpreted programming language that is freely available for many operating systems. It uses dynamic SQL, and it is ideal for prototyping applications.

Perl provides a standard module called the Database Interface (DBI) module for accessing different databases. It is available from <http://www.perl.com>. This module "talks" to drivers from different database vendors. In the case of DB2, this is the DBD::DB2 driver which is available from <http://www.ibm.com/software/data/db2/perl>.

### 1.3.10 Python

Python is a dynamic language often used for scripting. It emphasizes code readability and supports a variety of programming paradigms, including procedural, object-oriented, aspect-oriented, meta, and functional programming. Python is ideal for rapid application development.

Table 1.4 shows the extensions that are available for accessing DB2 databases from a Python application.

Extension	Description
<b>ibm_db</b>	Defined by IBM Provides the best support for advanced features. Allows you to issue SQL queries, call stored procedures, use pureXML®, and access metadata information.
<b>ibm_db_dbi</b>	Implements the Python Database API Specification v2.0. It does not offer some of the advanced features that the ibm_db API supports. If you have an application with a driver that supports Python Database API Specification v2.0, you can easily switch to ibm_db. The ibm_db and ibm_db_dbi APIs are packaged together.
<b>ibm_db_sa</b>	Supports SQLAlchemy, a popular open source Python SQL toolkit and object-to-relational mapper (ORM).

**Table 1.4 - IBM Data Server - Python extensions**

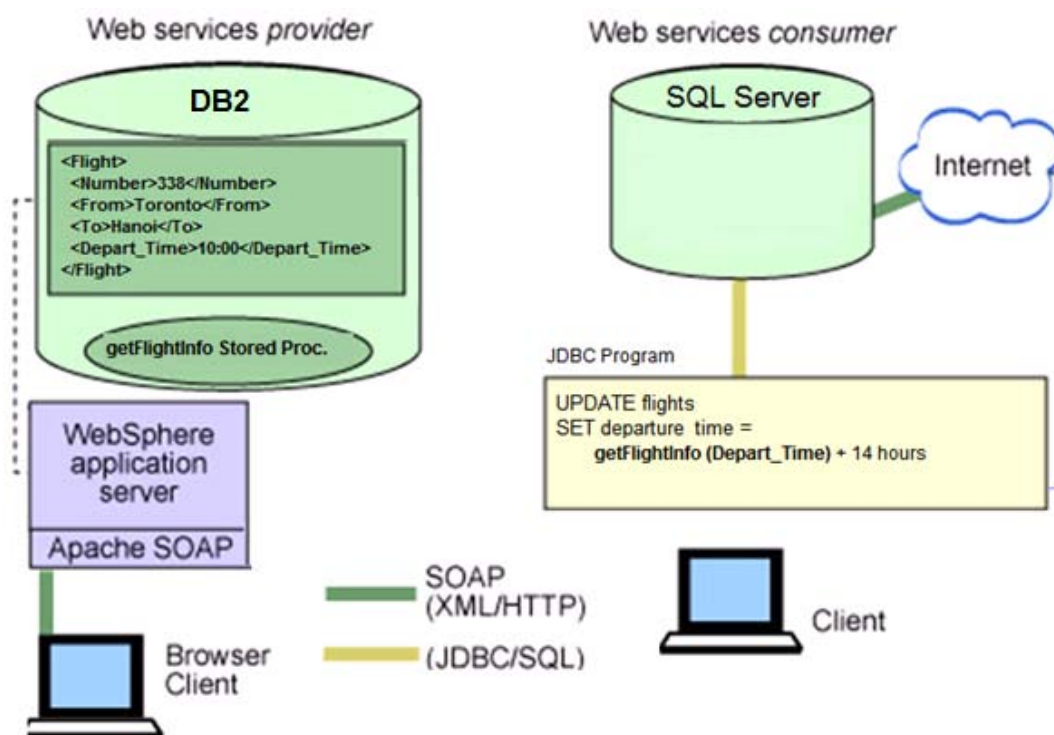
## 1.4 XML and DB2 pureXML

**Extensible Markup Language (XML)** is the underlying technology for Web 2.0 tools and techniques, as well as for **Service Oriented Architecture (SOA)**. IBM recognized early on the importance of XML, and large investments were made to deliver pureXML® technology -- a technology that provides for better storage support XML documents in DB2 software.

Introduced in 2006, DB2 9 is a hybrid data server: it allows native storage of relational data, as well as hierarchical data. While previous versions of DB2 and other data servers in the marketplace could store XML documents, the storage method used in DB2 9 has improved performance and flexibility. With DB2 9's pureXML technology, XML documents are stored internally in a parsed hierarchical manner, as a tree; therefore, working with XML documents is greatly enhanced. Newer releases of DB2 such as DB2 9.5 and DB2 9.7 have further improved the support for pureXML. *Chapter 15, DB2 pureXML* is devoted to this subject in detail.

## 1.5 Web services

As a simple definition, think of a Web service as a function you can invoke through the network, where you don't need to know the programming language used to develop it, you don't need to know the operating system where the function will run, and you don't need to know the location where it will run. Web services allow one application to exchange data with another application using extensible industry standard protocols based on XML. This is illustrated in *Figure 1.8*.



**Figure 1.8 – How an example Web service works**

In the figure, let's say the left side represents the system of a fictitious airline, Air Atlantis which is using DB2 on Linux, and stores its flight information in XML format in the DB2 database. On the right side we have a system from another fictitious airline, Air Discovery which is using SQL Server running on Windows. Now let's say that Air Atlantis and Air Discovery sign a partnership agreement where the two companies want to share scheduling and pricing information in order to coordinate their flights. Sharing information between the two may be a challenge given that the two companies are using different operating systems (Linux, Windows), and different data servers (DB2, SQL Server). When Air Atlantis changes its flight schedule for a trip going from Toronto to Beijing, how can Air Discovery automatically adjust its own flight schedule for a connecting flight from Beijing to Shanghai? The answer lies on Web services. Air Atlantis can expose some of its flight information by creating a **Data Web service** that returns the output of a stored procedure



(the *getFlightInfo* stored procedure) with flight information from the DB2 database. A **Data** Web service is a Web service based on database information. When this Data Web service is deployed to an application server such as WebSphere Application Server; then a client or partner like Air Discovery can use a browser to access Air Atlantis' flight information very easily. In this example, Air Atlantis behaves as the Web service provider as it developed and made available the Web service, while Air Discovery behaves as the Web service consumer since it consumes or uses the Web service.

Air Discovery can also invoke the Web service from its own JDBC application so that it can perform calculations that use data from its SQL Server database. For example, if a flight from Toronto to Beijing takes an average of 12 hours, Air Discovery can compute the connecting flight from Beijing to Shanghai by adding the departure time the Air Atlantis flight left Toronto, and adding the flight duration plus a few buffer hours. The amount of hours to use as buffer may be stored in the SQL Server database at Air Discovery's system, and the simple equation to use in the JDBC application would look like this:

Air Discovery Flight Departure time (Beijing to Shanghai)	=	Air Atlantis Flight Departure time (Toronto to Beijing)	+	Air Atlantis Flight Duration (Toronto to Beijing) 12 hours average	+	Air Atlantis Flight Buffer time (Toronto to Beijing) 2 hours
---	---	---	---	---	---	---

If Air Atlantis changes its flight departure time, this information is automatically communicated to the Air Discovery system when it invokes the Web service.

## 1.6 Administrative APIs

DB2 software ("DB2") provides a large amount of administrative APIs that developers can use to build their own utilities or tools. For example, to create a database you can invoke the *sqlcrea* API; to start an instance, use the *db2InstanceStart* API; or to import data into a table, use the *db2Import* API. The complete list is available from the DB2 Information Center. See the *Resources* section for the DB2 Information Center URL.

## 1.7 Development tools

Microsoft Visual Studio and Eclipse are two of the most popular Integrated Development Environments (IDEs) used by developers today. Both IDEs work well with DB2 software ("DB2").

Some users of DB2 also interact with third party products such as MS Excel and MS Access to create simple forms that connect to DB2. In this section we describe how to work with these products and DB2 Express-C.

DB2 Express-C is available also on Mac OS X, so you can use DB2 natively to develop database applications on a Mac. This may be especially appealing to the RoR community who has embraced the Mac platform.

### 1.7.1 Visual Studio

For Microsoft Visual Studio, DB2 provides the IBM Database Add-ins for Visual Studio. This free separate installable add-in integrates DB2 tools menus into Visual Studio after installed. This way, a developer does not need to switch to other tools to work with DB2 databases. You can download the Add-ins from the DB2 Express-C Web site at <http://ibm.com/db2/express>. More information is provided in *Chapter 6, Application development with .NET*.

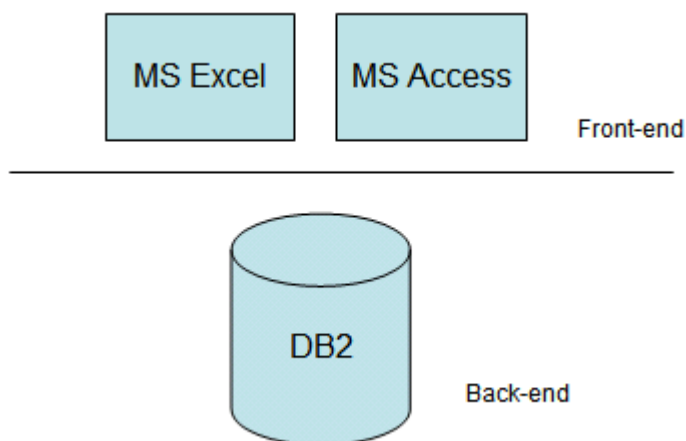
### 1.7.2 Eclipse

With respect to Eclipse, IBM offers IBM Data Studio, a free Eclipse-based tool that allows you to administer your DB2 instances and databases, and to develop SQL and XQuery scripts, stored procedures, UDFs, and data Web services. Because it is based on the Eclipse platform, many developers can leverage their existing knowledge to work with this tool.

To learn more about IBM Data Studio, refer to the eBook [Getting started with IBM Data Studio for DB2](#).

### 1.7.3 Access and Excel

Microsoft Excel and Microsoft Access are popular tools to generate reports, create forms, and develop simple applications that provide some business intelligence to your data. DB2 interacts very easily with these tools. A DBA can store the company data in a secure DB2 server, and regular users with Access or Excel can access this data and generate reports. This is illustrated in *Figure 1.9*



**Figure 1.9 - Working with Excel, Access and DB2**

In the figure, Excel and Access can be used to develop a front-end application, while DB2 takes care of data security, reliability and performance as the back-end of the application. Having all the data centralized in DB2 creates a simplified data storage model.

In the case of Excel, the easiest way to get access to the DB2 data is to use an OLE DB driver such as the IBM OLE DB Provider for DB2. This is included when you install the free IBM Data Server Client which can be downloaded from the DB2 Express-C web site at <http://ibm.com/db2/express>. Once installed, you need to select your data source with the appropriate OLE DB provider to use from the MS Excel menu. Choose *Data* → *Import External Data* → *Import Data*. The next steps are documented in the article *IBM® DB2® Universal Database™ and the Microsoft® Excel Application Developer... for Beginners* [1]. See the *References* section for details.

In the case of Microsoft Access, you should also have either of the following installed:

- IBM Data Server client, or
- IBM Data Server Driver for ODBC, CLI and .Net, or
- IBM Data Server Driver for ODBC and CLI

The IBM Data Server Driver for ODBC, CLI and .Net, and the IBM Data Server Driver for ODBC and CLI is also known as the IBM DB2 ODBC Driver, which is the same as the DB2 CLI driver. This is the driver to use to connect from Access to DB2. After the driver is installed, create an Access 2007 project, and choose the *ODBC Database* option available within the *External Data tab* in the *Table Tools* ribbon. The next steps are documented in the article *DB2 9 and Microsoft Access 2007 Part 1: Getting the Data...*[2]. When using **linked tables** in Microsoft Access, the data is available to Access 2007 users, but the data resides on the DB2 data server.

For versions of Access prior to 2007, the setup is a bit different, but you can review the article *Use Microsoft Access to interact with your DB2 data* [3]. See the *References* section for details.

## 1.8 Development environments

Developing applications using DB2 is not restricted to installing the software on your laptop, or company computer. Today you can take advantage of Cloud Computing to provision a DB2 server for a specified time. You can also develop DB2 applications using virtual appliances available for DB2. These development environments are discussed in this section.

### 1.8.1 DB2 Offerings on the Cloud

DB2 has offerings on the Cloud with:

- Amazon Web Services
- IBM development and test cloud
- IBM CloudBurst™ and IBM WebSphere CloudBurst appliance
- Rightscale™

These are discussed in the next sections in more detail.

### 1.8.1.1 Amazon Web Services

IBM has entered into a partnership agreement with Amazon Web Services (AWS) for running DB2 on Amazon's Elastic Compute Cloud (EC2). AWS delivers a set of integrated services that form a computing platform "in the cloud", and is available on a pay-as-you-go model. That is, AWS lets you 'rent' compute capacity (virtual servers and storage), and you only pay for the capacity that you utilize. For example, let's say you provision one EC2 virtual server for normal database operations, and during peak times or for seasonal needs you provision an extra database server for a few hours. In this example you would pay AWS only for the extra database server only for the few hours that you have it running.

IBM offers three different deployment options for DB2 on Amazon's cloud platform:

- DB2 Express-C Amazon Machine Images (AMIs) for evaluation and development
- Pay-as-you-go Production-ready AMIs with DB2 Express and DB2 Workgroup
- Ability to create your own AMIs using DB2 licenses you own

For more information and how to get started with DB2 on Amazon EC2, visit: <http://www.ibm.com/db2/cloud>

### 1.8.1.2 IBM development and test cloud

IBM Smart Business Development and Test on the IBM Cloud (IBM Developer Cloud for short) provides similar services to AWS, but it focuses on development and test. It allows for flexible provisioning of resources, on demand, at a predetermined cost.

At the time of writing, DB2 images on the IBM Developer Cloud include:

- DB2 Express-C 9.7.1 PAYG (Pay as you go).  
This image uses DB2 Express-C 9.7.1 built on 32-bit SUSE Linux Enterprise Server (SLES). Note that using DB2 Express-C is free; however, you need to pay for the infrastructure.
- DB2 Enterprise Developer 9.7.1 - BYOL (Bring your own license).  
This image uses DB2 Enterprise built on 32-bit SLES with the IBM Database Enterprise Developer Edition (DEDE) license.
- DB2 Enterprise Developer 9.7.1 64-bit - BYOL.  
This image uses DB2 Enterprise on 64-bit Red Hat Enterprise Linux (RHEL) with the IBM Database Enterprise Developer Edition (DEDE) license.

For more information about the IBM Developer Cloud, visit <http://ibm.com/cloud/enterprise>

### 1.8.1.3 IBM CloudBurst for development and test

IBM CloudBurst allows you to build your own private cloud in your company. It provides pre-installed, fully integrated service management capabilities across hardware, middleware and applications using the IBM System x® BladeCenter® platform. It includes services from IBM to implement it. Use IBM CloudBurst with the IBM WebSphere CloudBurst appliance. These two critical offerings complement each other to help your clients more easily, quickly and cost-effectively.

For more information, visit <http://www-01.ibm.com/software/tivoli/products/cloudburst/>

### 1.8.1.4 IBM WebSphere CloudBurst

IBM WebSphere CloudBurst is an appliance that helps developers establish and deploy software images and patterns into a cloud environment. WebSphere Cloudburst Appliance is like the “dispenser” of software environments into a private cloud, and IBM CloudBurst is the “recipient” private cloud environment.

At the time of writing the DB2 images available on the IBM Websphere CloudBurst are:

- DB2 Enterprise 9.7.0 32-bit trial with 90-day evaluation period

This image uses DB2 Enterprise built on 32-bit SLES

For more information, visit <http://www-01.ibm.com/software/websevers/cloudburst/>

### 1.8.1.4 Rightscale

RightScale is a Cloud Management Platform. It allows you to more easily deploy and manage business-critical applications on the cloud with automation, control, and portability. Rightscale provides server templates and scripts (called RightScripts) that are published on their site and allow you to automate, clone and repeat operations easily.

At the time of writing, the templates and Rightscripts listed in *Table 1.5* are available for DB2.

Type	Name	Description
ServerTemplate	IBM DB2 Express-C 9.7 (CentOS 5.2)	Install and configure DB2 Express-C 9.7
ServerTemplate	IBM DB2 Express-C 9.7 (Ubuntu 8.04)	Install and configure DB2 Express-C 9.7
RightScript	VPN Cubed Client Connect	Add server to existing VPN
RightScript	Add Users to Group	Add users to the group

RightScript	Backup DB2 Database	Run a DB2 backup on your running instance
RightScript	Create database	Create a database
RightScript	Create Group	Create a new group
RightScript	Create sample database	Run db2sampl
RightScript	Create User	Create a user
RightScript	Delete Group	Delete a system group
RightScript	Delete User	Delete a system user
RightScript	Drop a database	Drop a database
RightScript	Install DB2 Express-C	Install DB2 Express-C 9.7
RightScript	Remove Users from a Group	Remove users from group
RightScript	Run command within db2	Run command from db2 commandline
RightScript	Set DB2 parameter	Use the db2set command
RightScript	Start DB2	Start DB2 by running db2start
RightScript	Start DB2 Administration Server	Start DAS server by running db2admin start
RightScript	Stop DB2	Run "db2stop" command to stop db2
RightScript	Stop DB2 Administration Server	Stop DAS server by running db2admin stop

**Table 1.5 - Rightscale templates and RightScripts for DB2**

For more information about Rightscale and DB2, visit

[http://support.rightscale.com/27-Partners/IBM\\_DB2](http://support.rightscale.com/27-Partners/IBM_DB2)

## 1.8.2 DB2 Express-C virtual appliance for VMWare

If you work with VMWare in your company, you can use the DB2 Express-C virtual appliances for both, Linux and Windows on 32-bit or 64-bit. Download these appliances from <http://www.ibm.com/db2/express/download.html>

## 1.9 Sample programs

To help you learn how to program in different languages using DB2 as the data server, you can review the sample applications that come with the DB2 server installation in the SQLLIB\samples directory. *Figure 1.10* below shows some sample programs provided with DB2 on a Windows platform.

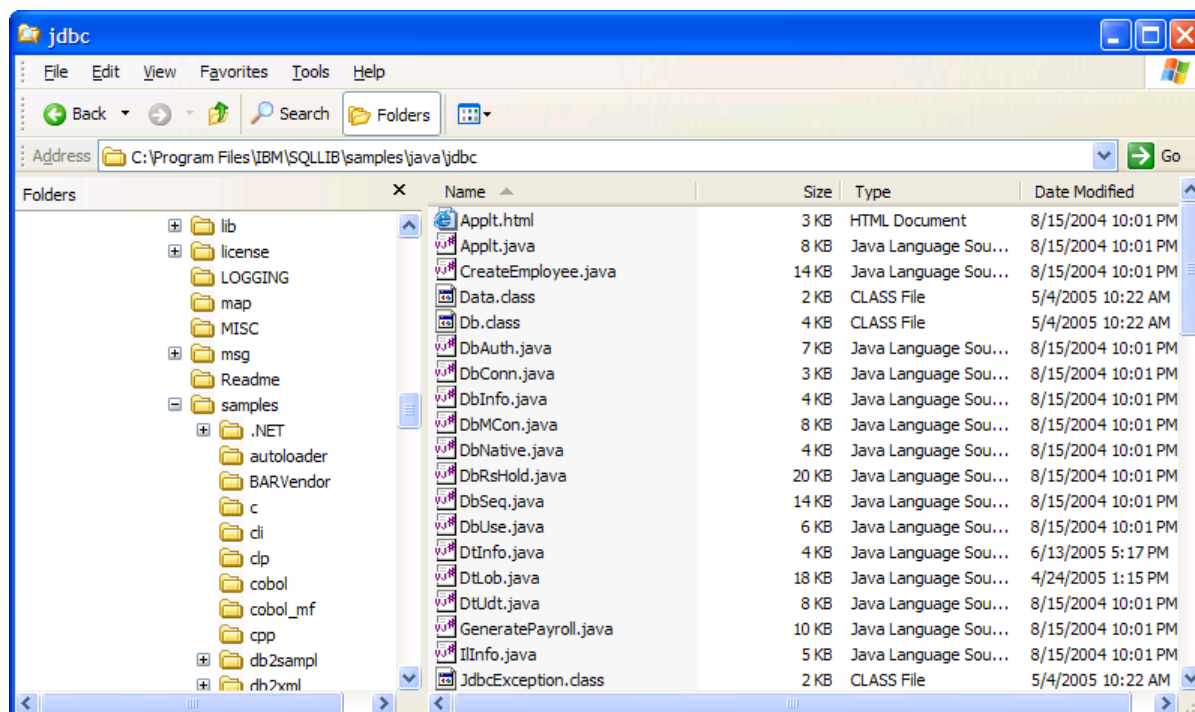


Figure 1.10 - Sample programs that come with DB2

## 1.10 Exercises

1. Using IBM Data Studio, create a stored procedure P2 that does nothing. Call this stored procedure from the DB2 Command Window or Linux shell.
2. Using IBM Data Studio, create a function F2 that takes no parameters and returns the value 2000. Invoke this function from the DB2 Command Window or Linux shell.
3. Modify P2 so it now invokes F2. Test it out!

## 1.11 Summary

In this chapter, we looked at how DB2 provides the flexibility to program database applications either inside the database on the server, or via client side applications with connections to the DB2 data server.

The server side application coverage included stored procedures, user-defined functions and triggers.

On the client side, we discussed the myriad of programming interfaces and methods permitted by DB2 application development, once again displaying the remarkable flexibility and capacity of DB2 as a database server.

## 1.12 Review questions

1. List one advantage of using stored procedures
2. How can a user extend the SQL language?
3. What is the difference between CLI and ODBC?
4. What is the difference between static SQL and dynamic SQL?
5. Mention one difference between a JDBC Type 2 and a JDBC Type 4 driver?
6. Which of the following objects are stored in a DB2 database?
  - A. Tables
  - B. Stored procedures
  - C. User-defined functions
  - D. All of the above
  - E. None of the above
7. Which of the following languages can be used to code stored procedures in DB2?
  - A. SQL PL
  - B. PL/SQL
  - C. Cobol
  - D. Java
  - E. All of the above
8. Choose the statements that are correct:
  - A. Static SQL is normally faster than dynamic SQL when the database is constantly changed with many updates, deletes and inserts.
  - B. Embedded SQL must be static SQL



- C. ODBC and JDBC always use dynamic SQL
  - D. All of the above
  - E. None of the above
9. Which of the following is the recommended ADO.NET provider to use with DB2?
- A. ODBC .NET Data provider
  - B. OLE DB .NET Data provider
  - C. DB2 .NET Data provider
  - D. All of the above
  - E. None of the above
10. Which of the following is not an IBM Data Server Python extension?
- A. ibm\_db
  - B. ibm\_db\_dbi
  - C. ibm\_db\_sa
  - D. ibm\_db\_python
  - E. All of the above



# 2

## Chapter 2 – DB2 pureXML

In this chapter we discuss pureXML, the new technology introduced in DB2 9 to support XML native storage. Many of the examples and concepts discussed in this chapter have been taken from the IBM Redbook®: *DB2 9: pureXML overview and fast start*. See the Resources section for more information on this title. *Figure 2.1* outlines which section of the DB2 “Big Picture” we discuss in this chapter.

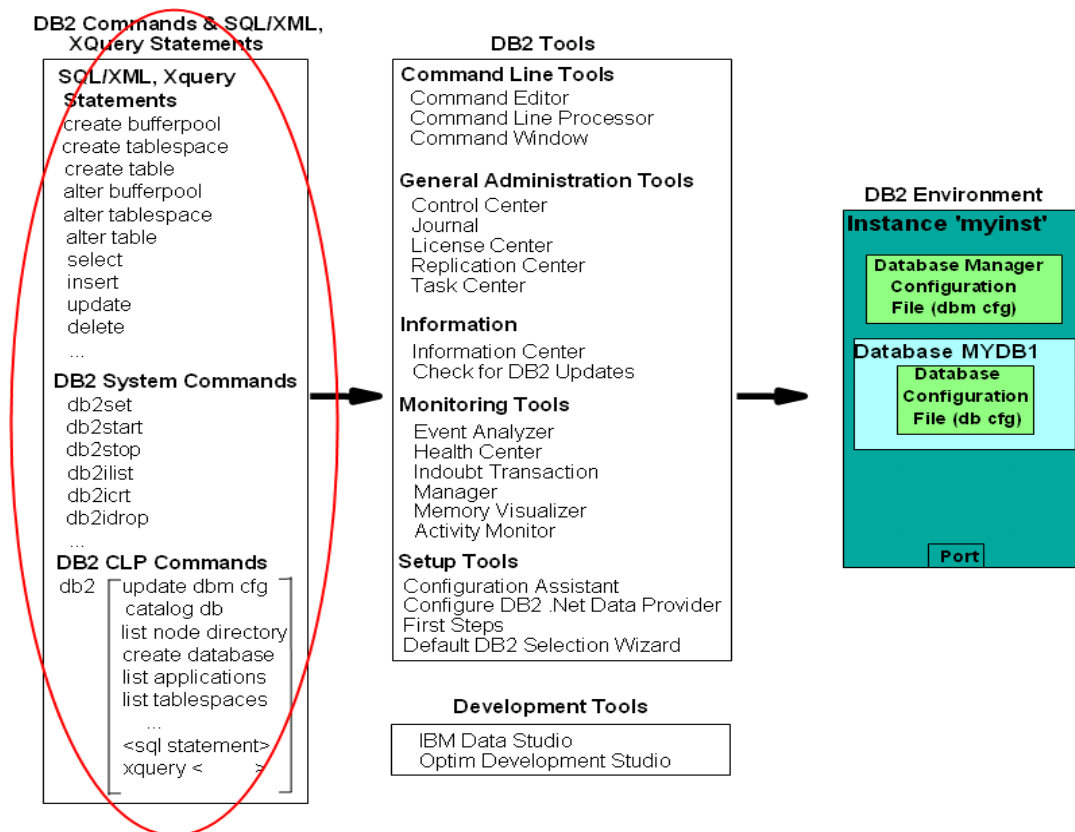


Figure 2.1 – The DB2 big picture: DB2 commands, SQL/XML and XQuery

**Note:**

For more information about pureXML, watch this video:

<http://www.channeldb2.com/video/video/show?id=807741:Video:4382>

## 2.1 Using XML with databases

XML documents can be stored in text files, XML repositories, or databases. There are two main reasons why many companies opt to store them in databases:

- Managing large volumes of XML data is a database problem. XML is data like other data, just in a different overall format. The same reasons to store relational data on databases apply to XML data: Databases provide efficient search and retrieval, robust support for persistence of data, backup and recovery, transaction support, performance and scalability.
- Integration: By storing relational and XML documents together, you can integrate new XML data with existing relational data, and combine SQL with XPath or XQuery in one query. Moreover, relational data can be published as XML, and vice versa. Through integration, databases can better support Web applications, SOA, and Web services.

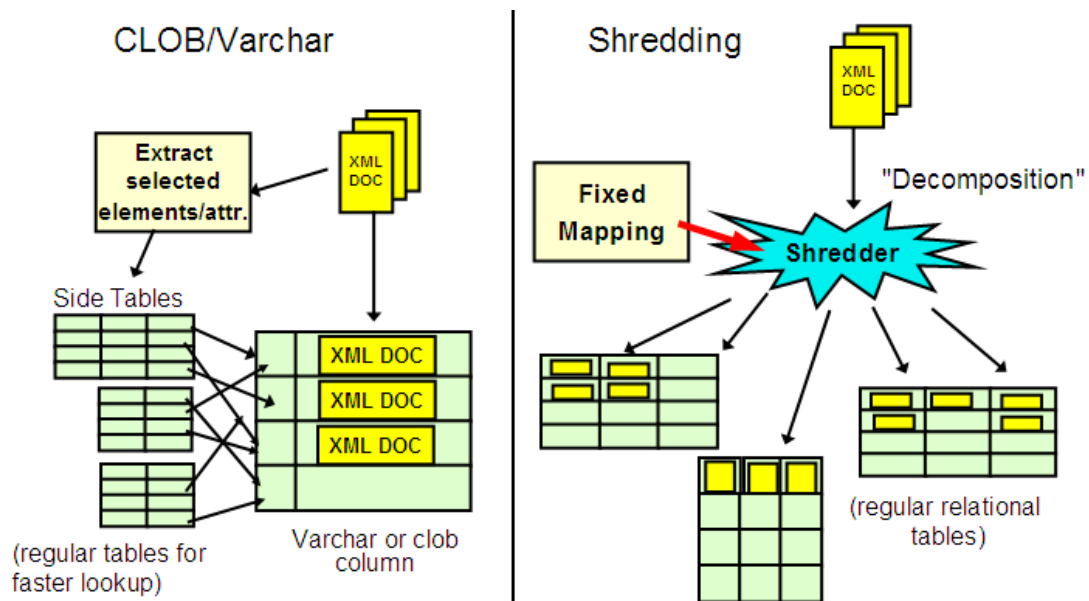
## 2.2 XML databases

There are two types of databases for storing XML data:

- XML-enabled databases
- Native XML databases

### 2.2.1 XML-enabled databases

An XML-enabled database uses the relational model as its core data storage model to store XML. This requires either a mapping between the XML (hierarchical) data model and the relational data model, or else storing the XML data as a character large object. While this can be considered as “old” technology, it is still being used by many database vendors. *Figure 2.2* explains in more detail the two options for XML-enabled databases.



**Figure 2.2 – Two options to store XML in XML-enabled databases**

The left side of *Figure 2.2* shows the **CLOB and varchar method** of storing XML documents in a database. Using this method, an XML document is stored as an unparsed string in either a CLOB or a varchar column in the database. If the XML document is stored as a string, when you want to retrieve part of the XML document, your program will have to retrieve the entire string, and parse it to find what you want. Think of parsing as building the XML document tree in memory so you can navigate through this tree. This method is memory-intensive and not very flexible.

The other option for XML-enabled databases is called **shredding or decomposition** and is illustrated on the right hand side of *Figure 2.2*. Using this method, an entire XML document is shredded into smaller parts which are stored in tables. Using this method, you are literally forcing an XML document, which is based on the hierarchical model, into the relational model. This method is not flexible because if the XML document is changed, this change is not easily propagated into the corresponding tables and many other tables may need to be created. This method is also not good for performance: if you need to get the original XML document back, you need to perform an expensive SQL join operation, which can become even more expensive when more tables are involved.

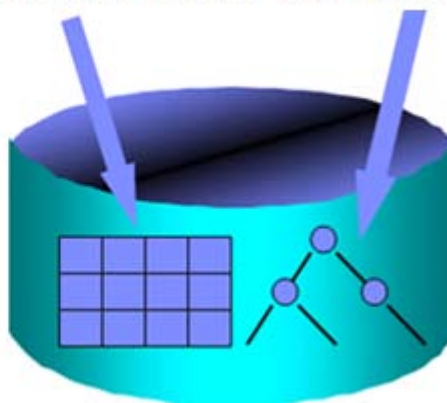
### 2.2.2 Native XML databases

Native XML databases use the hierarchical XML data model to store and process XML internally. The storage format is the same as the processing format: there is no mapping to the relational model, and XML documents are not stored as unparsed strings (CLOBs or varchars). When XPath or XQuery statements are used, they are processed natively by the engine, and not converted to SQL. This is why these databases are known as “native” XML databases. DB2 is currently the only commercial data server providing this support.

## 2.3 XML in DB2

Figure 2.3 below outlines how relational data and hierarchical data (XML documents) are both stored in a DB2 hybrid database. The figure also shows the **CREATE TABLE** statement that was used to create the table dept.

```
CREATE TABLE dept (deptID CHAR(8),..., deptdoc XML);
```

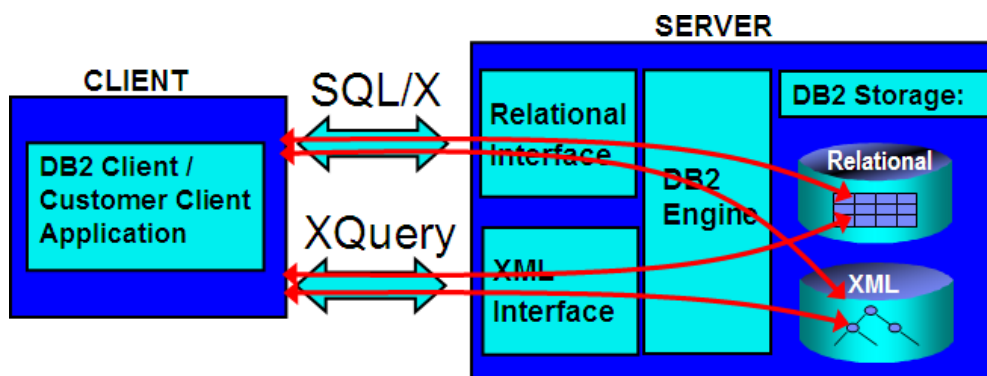


**Figure 2.3 – XML in DB2**

Note that the table definition uses a new data type, XML, for the deptdoc column. The left arrow in the figure indicates the relational column deptID stored in relational format (tables), while the XML column deptdoc is stored in parsed hierarchical format.

Figure 2.4 illustrates that in DB2 9, there are now four ways to access data:

- Use SQL to access relational data
- Use SQL with XML extensions (SQL/XML) to access XML data
- Use XQuery to access XML data
- Use XQuery to access relational data



**Figure 2.4 – Four ways to access data in DB2**

Thus, depending on your background, if you are an SQL person you may see DB2 as a world class RDBMS that also supports XML. If you are an XML person, you would see DB2 as a world class XML repository that also supports SQL.

Note that IBM uses the term *pureXML* instead of *native XML* to describe this technology. While other vendors still use the old technologies of CLOB/varchar or shredding to store XML documents, they call those old technologies “native XML”. To avoid confusion, IBM decided to use the new term pureXML, and to trademark this name so that no other database or XML vendor could use this same term to denote some differing technology. pureXML support is provided for databases created as Unicode or non-Unicode.

### 2.3.1 pureXML technology advantages

Many advantages are provided by pureXML technology.

1. You can seamlessly leverage your relational investment, given that XML documents are stored in columns of tables using the new XML data type.
2. You can reduce code complexity. For example, *Figure 2.5* illustrates a PHP script written with and without using pureXML. Using pureXML (the smaller box on the left side) the lines of code are reduced. This not only means that the code is less complex, but the overall performance is improved as there are fewer lines to parse and maintain in the code.

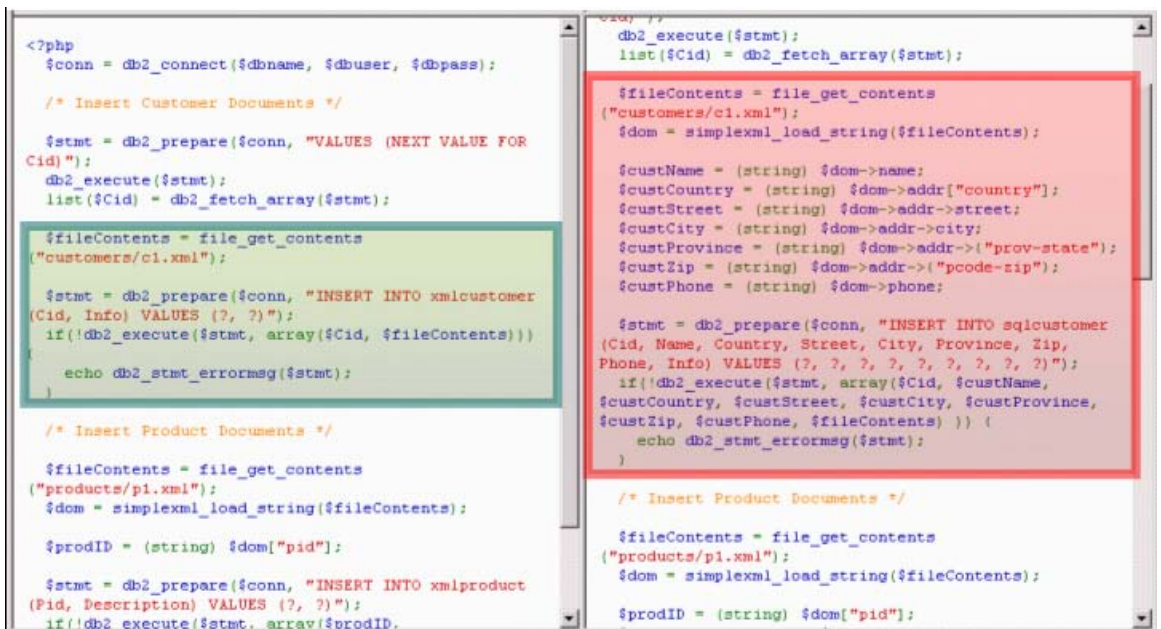


Figure 2.5 – Code complexity with and without pureXML

3. Changes to your schema are easier using XML and pureXML technology. *Figure 2.6* illustrates an example of this increased flexibility. In the figure, assume that you had a database consisting of the tables *Employee* and *Department*. Typically with a non-XML database, if your manager asked you to store not only one phone number per employee (the home phone number), but also a second phone number (a cell phone number), then you could add an extra column to the *Employee* table and store the cell phone number in that new column. However, this method would be against the normalization rules of relational databases. If you want to preserve these rules, you should instead create a new *Phone* side table, and move all phone information to this table. You could then also add the cell phone numbers as well. Creating a new *Phone* table is costly, not only because large amounts of pre-existing data needs to be moved, but also because all the SQL in your applications would have to change to point to the new table.

Instead, on the left side of the figure, we show how this could be done using XML. If employee *Christine* also has a cell phone number, a new tag can be added to put this information. If employee *Michael* does not have a cell phone number, we just leave it as is.

```
<DEPARTMENT deptid="15" deptname="Sales">
  <EMPLOYEE>
    <EMPNO>10</EMPNO>
    <FIRSTNAME>CHRISTINE</FIRSTNAME>
    <LASTNAME>SMITH</LASTNAME>
    <PHONE>408-463-4963</PHONE>
    <PHONE>415-010-1234</PHONE>
    <SALARY>52750.00</SALARY>
  </EMPLOYEE>
  <EMPLOYEE>
    <EMPNO>27</EMPNO>
    <FIRSTNAME>MICHAEL</FIRSTNAME>
    <LASTNAME>THOMPSON</LASTNAME>
    <PHONE>406-463-1234</PHONE>
    <SALARY>41250.00</SALARY>
  </EMPLOYEE>
</DEPARTMENT>
```

#### Requires:

- Normalization of existing data !
- Modification of the mapping
- Change of applications

#### Phone

EMPNO	PHONE
27	406-463-1234
10	415-010-1234
10	408-463-4963

#### Department

DEPTID	DEPTNAME
15	Sales

Costly!

#### Employee

DEPTID	EMPNO	FIRSTNAME	LASTNAME	PHONE	SALARY
15	27	MICHAEL	THOMPSON	406-463-1234	41250
15	10	CHRISTINE	SMITH	408-463-4963	52750

**Figure 2.6 – Increased data flexibility using XML**

4. You can improve your XML application performance. Tests performed using pureXML technology showed huge improvements in performance for XML applications. *Table 2.1* shows the test results for a company that switched to pureXML from older technologies. The second column shows the results using the old method of working with XML using another relational database, and the third column shows the results using DB2 with pureXML.



Task	Other relational DB	DB2 with pureXML
Development of search and retrieval business processes	CLOB: 8 hrs Shred: 2 hrs	30 min.
Relative lines of I/O code	100	35 (65% reduction)
Add field to schema	1 week	5 min.
Queries	24 - 36 hrs	20 sec - 10 min

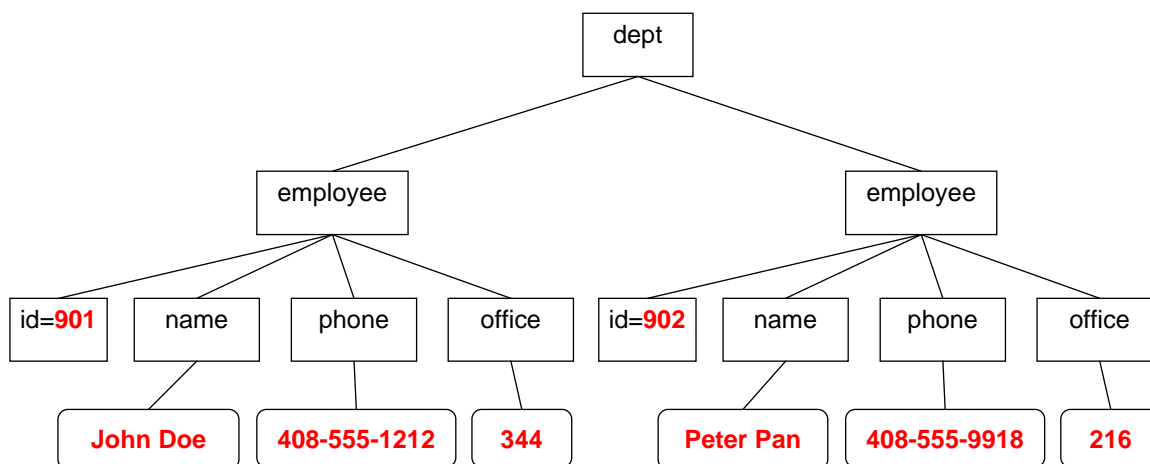
**Table 2.1 – Increased performance using pureXML technology**

### 2.3.2 XPath basics

XPath is a language that can be used to query XML documents. *Listing 2.1* shows an XML document, and *Figure 2.7* illustrates the same document represented in parsed-hierarchical (also called “node” or “leaf”) format. We will use the parsed-hierarchical format to explain how XPath works.

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
  <phone>408 555 1212</phone>
  <office>344</office>
</employee>
<employee id="902">
  <name>Peter Pan</name>
  <phone>408 555 9918</phone>
  <office>216</office>
</employee>
</dept>
```

**Listing 2.1 – An XML document**



**Figure 2.7 – Parsed-hierarchical representation of the XML document in Listing 2.1**

A quick way to learn XPath is to compare it to the change directory (`cd`) command in MS-DOS or Linux/UNIX. Using the `cd` command, you traverse a directory tree as follows:

```
cd /directory1/directory2/...
```

Similarly, in XPath you use slashes to go from one element to another within the XML document. For example, using the document in *Listing 2.1* in XPath you could retrieve the names of all employees using this query:

```
/dept/employee/name
```

### 2.3.2.1 XPath expressions

XPath expressions use fully qualified paths to specify elements and attributes. An “@” sign is used to specify an attribute. To retrieve only the value (text node) of an element, use the `text()` function. *Table 2.2* shows XPath queries and the corresponding results using the XML document from *Listing 2.1*.

XPath	Result
<code>/dept/@bldg</code>	101
<code>/dept/employee/@id</code>	901 902
<code>/dept/employee/name</code>	<name>Peter Pan</name> <name>John Doe</name>

/dept/employee/name/text()	Peter Pan John Doe
----------------------------	-----------------------

**Table 2.2 – XPath expression examples**

### 2.3.2.2 XPath wildcards

There are two main wildcards in XPath:

- “\*” matches any tag name
- “//” is the “descendent-or-self” wildcard

Table 2.3 provides more examples using the XML document from *Listing 2.1*

XPath	Result
/dept/employee/*/text()	John Doe 408 555 1212 344 Peter Pan 408 555 9918 216
/dept/*/@id	901 902
//name/text()	Peter Pan John Doe
/dept//phone	<phone>408 555 1212</phone> <phone>408 555 9918</phone>

**Table 2.3 – XPath wildcard examples**

### 2.3.2.3 XPath predicates

Predicates are enclosed in square brackets [ ]. As an analogy, you can think of them as the equivalent to the WHERE clause in SQL. For example [ @id="902" ] can be read as: “WHERE attribute id is equal to 902”. There can be multiple predicates in one XPath expression. To specify a positional predicate, use [n] which means the n<sup>th</sup> child would be selected. For Example, employee[2] means that the second employee should be selected. Table 2.4 provides more examples.

XPath	Result
/dept/employee[@id="902"]/name	<name>Peter Pan</name>
/dept[@bldg="101"]/employee[office >"300"]/name	<name>John Doe</name>

//employee[office="344" office="216"]/@id	OR	901 902
/dept/employee[2]/@id		902

**Table 2.4 – XPath predicate examples**

### 2.3.2.4 The parent axis

Similar to MS-DOS or Linux/UNIX, you can use a "." (dot) to indicate in the expression that you are referring to the current context, and a ".." (dot dot) to refer to the parent context. *Table 2.5* provides more examples.

XPath	Result
/dept/employee/name[../@id="902"]	<name>Peter Pan</name>
/dept/employee/office[.>"300"]	<office>344</office>
/dept/employee[office > "300"]/office	<office>344</office>
/dept/employee[name="John Doe"]/../@bldg	101
/dept/employee/name[.="John Doe"]/../../@bldg	101

**Table 2.5 – XPath parent axis**

### 2.3.3 XQuery basics

XQuery is a query language created for XML. XQuery supports path expressions to navigate the XML hierarchical structure. In fact, XPath is a subset of XQuery; therefore, everything we learned earlier about XPath applies to XQuery too. XQuery supports both typed and untyped data. XQuery lacks null values because XML documents omit missing or unknown data. XQuery and XPath expressions are case sensitive, and XQuery returns sequences of XML data.

XQuery supports the FLWOR expression. If we use SQL for an analogy, it is equivalent to a SELECT-FROM-WHERE expression. The next section describes FLWOR in more detail.

#### 2.3.3.1 XQuery: FLWOR expression

FLWOR stands for:

- FOR: iterates through a sequence, binds a variable to items
- LET: binds a variable to a sequence
- WHERE: eliminates items of the iteration
- ORDER: reorders items of the iteration

- RETURN: constructs query results

It is an expression that allows manipulation of XML documents, enabling you to return another expression. For example, assume you have a table with this definition:

```
CREATE TABLE dept(deptID CHAR(8),deptdoc XML);
```

And the XML document in *Listing 2.2* is inserted in the *deptdoc* column

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

#### Listing 2.2 - A sample XML document

Then the XQuery statement in *Listing 2.3* using the FLWOR expression could be run:

```
xquery
for $d in db2-fn:xmlcolumn('dept.deptdoc')/dept
let $emp := $d//employee/name
where $d/@bldg > 95
order by $d/@bldg
return
  <EmpList>
    {$d/@bldg, $emp}
  </EmpList>
```

#### Listing 2.3 - A sample XQuery statement with the FLWOR expression

This would return the output shown in *Listing 2.4*

```
<EmpList bldg="101">
  <name>
    John Doe
  </name>
  <name>
    Peter Pan
  </name>
</EmpList>
```

#### Listing 2.4 - Output after running the XQuery statement in Listing 2.3

### 2.3.4 Inserting XML documents

Inserting XML documents into a DB2 database can be performed using the SQL INSERT statement, or the IMPORT utility. XQuery cannot be used for this purpose as this has not yet been defined in the standard.

Let's examine the script `table_creation.txt` shown in *Listing 2.5* below, which can be run from the DB2 Command Window or Linux shell using this statement:

```
db2 -tvf table_creation.txt

-- (1)
drop database mydb
;

-- (2)
create database mydb using codeset UTF-8 territory US
;

-- (3)
connect to mydb
;

-- (4)
create table items (
    id            int primary key not null,
    brandname     varchar(30),
    itemname      varchar(30),
    sku           int,
    srp           decimal(7,2),
    comments      xml
);

-- (5)
create table clients(
    id            int primary key not null,
    name          varchar(50),
    status        varchar(10),
    contact       xml
);

-- (6)
insert into clients values (77, 'John Smith', 'Gold',
    '<addr>111 Main St., Dallas, TX, 00112</addr>')
;
```

```
-- (7)
IMPORT FROM "D:\Raul\clients.del" of del xml from "D:\Raul" INSERT INTO
CLIENTS (ID, NAME, STATUS, CONTACT)
;

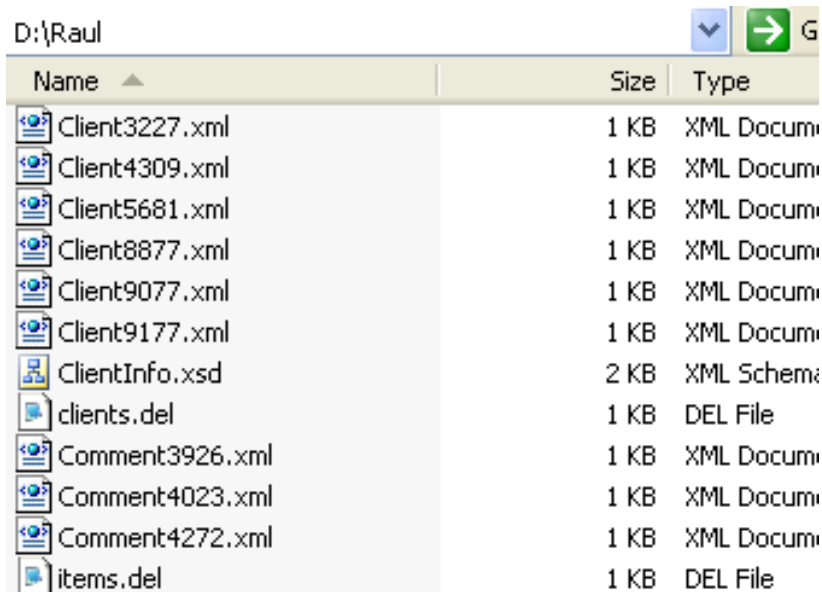
-- (8)
IMPORT FROM "D:\Raul\items.del" of del xml from "D:\Raul" INSERT INTO
ITEMS (ID, BRANDNAME, ITEMNAME, SKU, SRP, COMMENTS)
;
```

### Listing 2.5 - Contents of the file `table_creation.txt`

Note that this script file and related files are provided in the compressed file **Exercise\_Files\_DB2\_Application\_Development.zip** that accompanies this book. Follow along as we describe each line in the script of *Listing 2.5*.

1. Drop the database *mydb*. This is normally done in script files to perform cleanup. If *mydb* didn't exist before, you will receive an error message, but this is OK.
2. Create the database *mydb* using the codeset UTF-8. This creates a Unicode database. pureXML is supported in both Unicode and non-Unicode databases.
3. Connect to the newly created database *mydb*. This is necessary to create objects within the database.
4. Create the table *items*. Note that the last column in the table (column *comments*) is defined as an XML column using the new XML data type.
5. We create the table *clients*. Note that the last column in the table (column *contact*) is also defined with the new XML data type.
6. Using this SQL INSERT statement, you can insert an XML document into an XML column. In the INSERT statement you pass the XML document as a string enclosed in single quotes.
7. Using an IMPORT command, you can insert or import several XML documents along relational data into the database. In (7) you are importing the data from the *clients.del* file (a delimited ascii file), and you also indicate where the XML data referenced by that *clients.del* file is located (for this example, in `D:\Raul`).

We will take a more careful look at file `clients.del`, but first, let's see the contents of directory `D:\Raul` first. *Figure 2.8* provides this information.



Name	Size	Type
Client3227.xml	1 KB	XML Document
Client4309.xml	1 KB	XML Document
Client5681.xml	1 KB	XML Document
Client8877.xml	1 KB	XML Document
Client9077.xml	1 KB	XML Document
Client9177.xml	1 KB	XML Document
ClientInfo.xsd	2 KB	XML Schema
clients.del	1 KB	DEL File
Comment3926.xml	1 KB	XML Document
Comment4023.xml	1 KB	XML Document
Comment4272.xml	1 KB	XML Document
items.del	1 KB	DEL File

**Figure 2.8 - Contents of D:\Raul directory with XML documents**

*Listing 2.6* shows the contents of the text file `clients.del`.

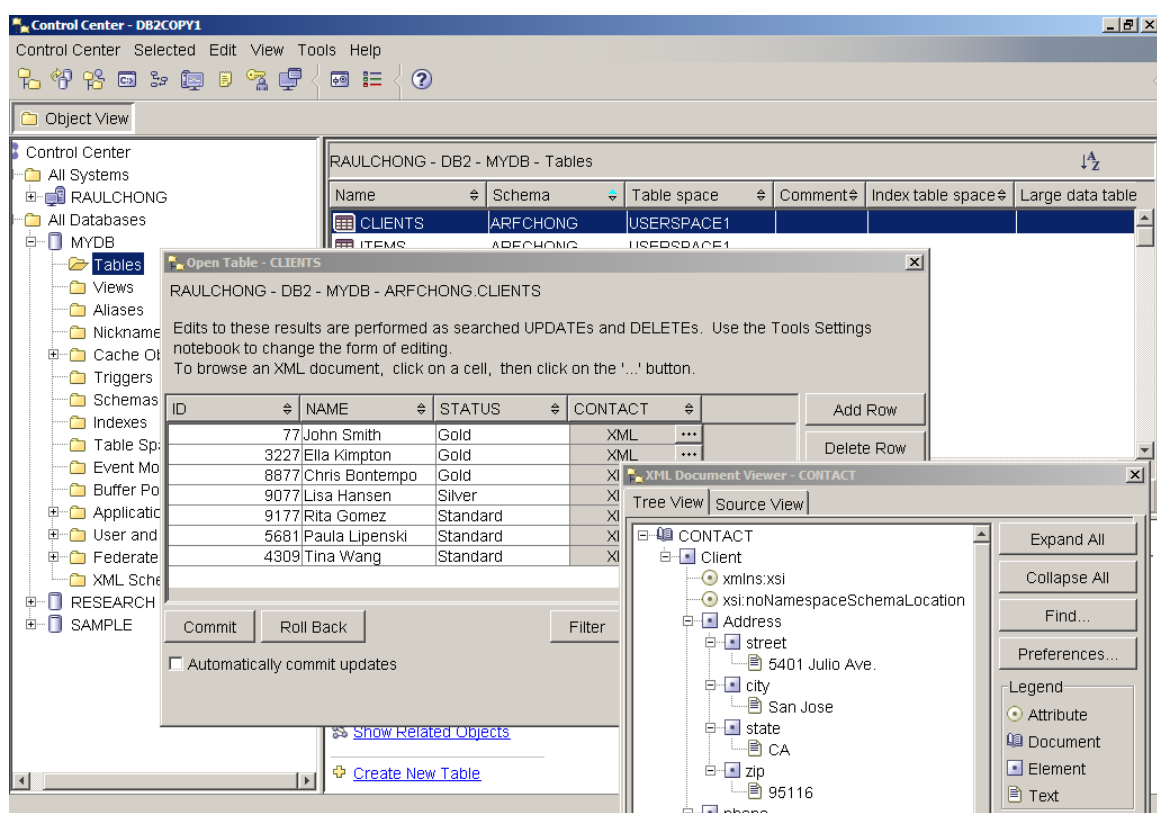
```
3227,Ella Kimpton,Gold,<XDS FIL='Client3227.xml' />,
8877,Chris Bontempo,Gold,<XDS FIL='Client8877.xml' />,
9077,Lisa Hansen,Silver,<XDS FIL='Client9077.xml' />
9177,Rita Gomez,Standard,<XDS FIL='Client9177.xml' />,
5681,Paula Lipenski,Standard,<XDS FIL='Client5681.xml' />,
4309,Tina Wang,Standard,<XDS FIL='Client4309.xml' />
```

**Listing 2.6 - Contents of the file `clients.del`**

In the `clients.del` file, “XDS FIL=” is used to point to a specific XML document file.

Figure 2.9 shows the Control Center after running the above script.





**Figure 2.9 – The Control Center after running the table\_creation.txt script**

Note that in the figure, we show the contents of the `CLIENTS` table. The last column `contact` is an XML column. When you click on the button with three dots, another window opens showing you the XML document contents. This is shown in the bottom right corner of the *Figure 2.9*.

### 2.3.5 Querying XML data

There are two ways to query XML data in DB2 software:

- Using SQL with XML extensions (SQL/XML)
- Using XQuery

In both cases, DB2 follows international XML standards.

#### 2.3.5.1 Querying XML data with SQL/XML

Using regular SQL statements allows you to work with rows and columns. An SQL statement can be used to work with full XML documents; however, it would not help when attempting to retrieve only part of the document. In such cases, you need to use SQL with XML extensions (SQL/XML).

Table 2.6 describes some of the SQL/XML functions available with the SQL 2006 standard

Function name	Description
XMLPARSE	Parses character or large object binary data, produces XML value
XMLSERIALIZE	Converts an XML value into character or large object binary data
XMLVALIDATE	Validates XML value against an XML schema and type-annotates the XML value
XMLEXISTS	Determines if an XQuery returns a results (i.e. a sequence of one or more items)
XMLQUERY	Executes an XQuery and returns the result sequence
XMLTABLE	Executes an XQuery, returns the result sequence as a relational table (if possible)
XMLCAST	Cast to or from an XML type

**Table 2.6 – SQL/XML Functions**

The following examples can be tested using the *mydb* database created earlier.

### Example 1

Imagine that you need to locate the names of all clients who live in a specific zip code. The `clients` table stores customer addresses, including zip codes, in an XML column. Using `XMLEXISTS`, you can search the XML column for the target zip code and then restrict the return result set accordingly. *Listing 2.7* below illustrates the query required.

```
SELECT name FROM clients
WHERE xmlexists(
    '$c/Client/Address[zip="95116"]'
    passing clients.contact as "c"
)
```

### Listing 2.7 - An example using XMLEXISTS

In *Listing 2.7*, the first line is an SQL clause specifying that you want to retrieve information in the `name` column of the `clients` table.

The `WHERE` clause invokes the `XMLEXISTS` function, specifying the XPath expression that prompts DB2 to navigate to the `zip` element and check for a value of 95116

The `$c/Client/Address` clause indicates the path inside the XML document hierarchy where DB2 can locate the `zip` element. A dollar sign (\$) is used to specify a variable;

therefore “c” is a variable. This variable is then defined by this line: `passing clients.contact as "c"`. Here, `clients` is the name of the table and `contact` is the name of the column with an XML data type. In other words, we are passing the XML document to the variable “c”.

DB2 inspects the XML data contained in the `contact` column, navigates from the root `Client` node down to the `Address` node, then to the `zip` node and finally determines if the customer lives in the target zip code. The `XMLEXISTS` function evaluates to “true” and DB2 returns the name of the client associated with that row.

Starting with DB2 9.5, the above query could be simplified as shown in *Listing 2.8* below.

```
SELECT name FROM clients
WHERE xmlexists(
  '$CONTACT/Client/Address[zip="95116"]'
)
```

#### **Listing 2.8 - Simplified version of the query shown in Listing 2.7**

A variable with the same name as an XML column is created automatically by DB2. In the above example, the variable `CONTACT` is created automatically by DB2. Its name matches the name of the XML column `CONTACT`.

#### **Example 2**

Let’s consider how to solve the problem of how to create a report listing the e-mail addresses of “Gold” status customers. The query in *Listing 2.9* below could be run for this purpose.

```
SELECT xmlquery('$c/Client/email' passing contact as "c")
FROM clients
WHERE status = 'Gold'
```

#### **Listing 2.9 - An example using XMLQUERY**

The first line indicates we want to return the email address which is an element of the XML document (not a relational column). As in the previous example, “\$c” is a variable that contains the XML document. In this example we use the `XMLQUERY` function which can be used after a `SELECT`, while the `XMLEXISTS` function can be used after a `WHERE` clause.

#### **Example 3**

There may be situations when you would like to present XML data as tables. This is possible with the `XMLTABLE` function as shown in *Listing 2.10* below.

```
SELECT t.comment#, i.itemname, t.customerID, Message
FROM items i,
xmltable('$c/Comments/Comment' passing i.comments as "c"
```

```
columns Comment# integer path 'CommentID',
CustomerID integer path 'CustomerID',
Message varchar(100) path 'Message') AS t
```

#### Listing 2.10 - An example using XMLTABLE

The first line specifies the columns to be included in your results set. Columns prefixed with the “t” variable are based on XML element values.

The third line invokes the XMLTABLE function to specify the DB2 XML column containing the target data (`i.comments`) and the path within the column's XML documents where the elements of interest are located.

The `columns` clause, spanning lines 4 to 6, identifies the specific XML elements that will be mapped to output columns in the SQL result set specified on line 1. Part of this mapping involves specifying the data types to which the XML element values will be converted. In this example all XML data is converted to traditional SQL data types.

#### Example 4

Now let's explore a simple example in which you include an XQuery FLWOR expression inside an XMLQUERY SQL/XML function. This is illustrated in *Listing 2.11*.

```
SELECT name, xmlquery(
  'for $e in $c/Client/email[1] return $e'
  passing contact as "c"
)
FROM clients
WHERE status = 'Gold'
```

#### Listing 2.11 - An example using XMLQUERY and FLWOR

The first line specifies that the customer names and the output from the XMLQUERY function will be included in the result set. The second line indicates the first `email` sub-element of the `Client` element is to be returned. The third line identifies the source of our XML data (`contact` column). The fourth line tells us that this column is coming from the `clients` table; and the fifth line indicates that only `Gold` customers are of interest.

#### Example 5

The example illustrated in *Listing 2.12* demonstrates again the XMLQUERY function which takes an XQuery FLWOR expression; however, note that this time we are returning not only XML, but also HTML.

```
SELECT xmlquery('for $e in $c/Client/email[1]/text()
  return <p>{$e}</p>'
  passing contact as "c")
FROM clients
```

```
WHERE status = 'Gold'
```

### Listing 2.12 - An example returning XML and HTML

The return clause of XQuery enables you to transform XML output as needed. Using the `text()` function in the first line indicates that only the text representation of the first email address of qualifying customers is of interest. The second line specifies that this information is to be surrounded by HTML paragraph tags.

### Example 6

The following example uses the `XMLEMENT` function to create a series of item elements, each of which contain sub-elements for the ID, brand name, and stock keeping unit (SKU) values obtained from corresponding columns in the `items` table. Basically, you can use the `XMLEMENT` function when you want to convert relational data to XML data. This is illustrated in *Listing 2.13*.

```
SELECT
  xmlement (name "item", itemname),
  xmlement (name "id", id),
  xmlement (name "brand", brandname),
  xmlement (name "sku", sku)
FROM items
WHERE srp < 100
```

### Listing 2.13 - An example using XMLEMENT

The query in *Listing 2.13* would return the output as shown in *Listing 2.14*

```
<item>
  <id>4272</id>
  <brand>Classy</brand>
  <sku>981140</sku>
</item>
...
<item>
  <id>1193</id>
  <brand>Natural</brand>
  <sku>557813</sku>
</item>
```

### Listing 2.14 - Output of the query in Listing 2.13

#### 2.3.5.2 Querying XML data with XQuery

In the previous section, we looked at how to query XML data using SQL with XML extensions. SQL was always the primary query method, and XPath or XQuery was embedded inside SQL. In this section, we discuss how to query XML data using XQuery.

This time, XQuery will be the primary query method, and in some cases, we will use SQL embedded inside XQuery (using the `db2-fn:sqlquery` function). When using XQuery, we will invoke a few functions, and will also use the FLWOR expression.

### Example 1

This is a simple XQuery to return customer contact data. In the example, `CONTACT` is the name of the XML column, and `CLIENTS` is the name of the table.

```
xquery db2-fn:xmlcolumn('CLIENTS.CONTACT')
```

Always prefix any XQuery expression with the `xquery` command so that DB2 knows it has to use the XQuery parser, otherwise DB2 will assume you are trying to run an SQL expression. The `db2-fn:xmlcolumn` function is a function that retrieves the XML documents from the column specified as the parameter. It is equivalent to the following SQL statement, as it is retrieving the entire column contents:

```
SELECT contact FROM clients
```

### Example 2

In this example shown in *Listing 2.15*, we use the FLWOR expression to retrieve client fax data

```
xquery
  for $y in db2-fn:xmlcolumn('CLIENTS.CONTACT')/Client/fax
  return $y
```

#### Listing 2.15 - XQuery and the FLWOR expression

The first line invokes the XQuery parser. The second line instructs DB2 to iterate through the fax sub-elements contained in the `CLIENTS.CONTACT` column. Each fax element is bound to the variable `$y`. The third line indicates that for each iteration, the value “`$y`” is returned.

The output of this query is illustrated in *Listing 2.16* (We omitted the namespace in the output, otherwise it would be harder to read as it may span several lines):

```
<fax>4081112222</fax>
<fax>5559998888</fax>
```

#### Listing 2.16 - Output of the query show in Listing 2.15

### Example 3

The example in Listing 2.17 queries XML data and returns the results as HTML.

```
xquery
  <ul> {
    for $y in db2-fn:xmlcolumn('CLIENTS.CONTACT')/Client/Address
```

```

    order by $y/zip
    return <li>{$y}</li>
  }
</ul>

```

**Listing 2.17 - XQuery statement with the FLWOR expression returning HTML**

The sample HTML returned would look as shown in *Listing 2.18*.

```

<ul>
<li>
<address>
  <street>9407 Los Gatos Blvd.</street>
  <city>Los Gatos</city>
  <state>ca</state>
  <zip>95302</zip>
</address>
</li>
<address>
<street>4209 El Camino Real</street>
  <city>Mountain View</city>
  <state>CA</state>
  <zip>95302</zip>
</address>
</li>
...
</ul>

```

**Listing 2.18 - Output of the query ran in Listing 2.17**

**Example 4**

The following example shows how to embed SQL within XQuery by using the `db2-fn:sqlquery` function. The `db2-fn:sqlquery` function executes an SQL query and returns only the selected XML data. The SQL query passed to `db2-fn:sqlquery` must only return XML data. This XML data can then be further processed by XQuery. This is illustrated in *Listing 2.19*.

```

xquery
  for $y in
    db2-fn:sqlquery(
      'select comments from items where srp > 100'
    )/Comments/Comment
  where $y/ResponseRequested='Yes'
  return (

```

```

    <action>
      {$y/ProductID
        $y/CustomerID
        $y/Message}
    </action>
  )

```

**Listing 2.19 - An example of the `db2-fn:sqlquery` function embedding SQL within XQuery**

In the example, the SQL query filters rows based on the condition that the `srp` column has a value greater than 100. From those rows filtered, it will pick the `comments` column, which is the XML column. Next XQuery (or XPath) is applied to go to sub-elements.

**Note:**

SQL is case insensitive and DB2 software stores all table and column names in uppercase by default. XQuery on the other hand, is case sensitive. The above functions are XQuery interface functions so all the table names and column names should be passed to these functions in uppercase. Passing the object names in lowercase may result in an undefined object name error.

### 2.3.6 Joins with SQL/XML

This section describes how to perform JOIN operations between two XML columns of different tables, or between one XML column and one relational column. Assume you have created two tables with the statements shown in *Listing 2.20*

```

CREATE TABLE dept (unitID CHAR(8), deptdoc XML)
CREATE TABLE unit (unitID CHAR(8) primary key not null,
                  name CHAR(20),
                  manager VARCHAR(20),
                  ...
                  )

```

**Listing 2.20 - DDL of tables to use in the JOIN examples**

You can perform a JOIN operation in either of two ways. The first method is shown in *Listing 2.21*.

```

SELECT u.unitID
FROM dept d, unit u
WHERE XMLEXISTS (
  '$e//employee[name = $m]'
  passing d.deptdoc as "e", u.manager as "m")

```

**Listing 2.21 - First method to perform a JOIN with SQL/XML**

Line 4 of the statement in the above listing shows that the JOIN operation occurs between the element `name`, which is a sub-element of the `deptdoc` XML column in table `dept`, and the `manager` relational column in the table `unit`.



*Listing 2.22* shows the second method to perform the JOIN.

```
SELECT u.unitID
FROM dept d, unit u
WHERE u.manager = XMLCAST(
  XMLQUERY('$e//employee/name '
    passing d.deptdoc as "e")
  AS char(20))
```

### Listing 2.22 - Second method to perform a JOIN with SQL/XML

In this second method, the relational column is on the left side of the JOIN. If the relational column is on the left side of the equation, a relational index may be used instead of an XML index.

## 2.3.7 Joins with XQuery

Assume the following tables have been created:

```
CREATE TABLE dept(unitID CHAR(8), deptdoc XML)
CREATE TABLE project(projectDoc XML)
```

If we use SQL/XML, a JOIN would look as shown in *Listing 2.23*.

```
SELECT XMLQUERY (
  '$d/dept/employee' passing d.deptdoc as "d")
FROM dept d, project p
WHERE XMLEXISTS (
  '$e/dept[@deptID=$p/project/deptID]'
  passing d.deptdoc as "e", p.projectDoc as "p")
```

### Listing 2.23 - A JOIN with SQL/XML

The equivalent JOIN using XQuery is shown in *Listing 2.24*.

```
xquery
for $dept in db2-fn:xmlcolumn("DEPT.DEPTDOC")/dept
for $proj in db2-fn:xmlcolumn("PROJECT.PROJECTDOC")/project
where $dept/@deptID = $proj/deptID
return $dept/employee
```

### Listing 2.24 - A JOIN with XQuery

This second method is easier to interpret -- variable `$dept` holds the XML document of the XML column `deptdoc` in table `dept`. The variable `$proj` holds the XML document of the XML column `projectdoc` in table `project`. Then line 4 performs the JOIN operation between an attribute of the first XML document and an element of the second XML document.

### 2.3.8 Update and delete operations

Update and delete operations on XML data can be performed in one of two ways:

- Using SQL UPDATE and DELETE statements
- Using the TRANSFORM expression

For the first way using SQL UPDATE and DELETE statements, the update or delete occurs at the document level; that is, the entire XML document is replaced with the updated one. For example, in the UPDATE statement in *Listing 2.25* below, if you'd only like to change the <state> element, the entire XML document is actually replaced.

```
UPDATE clients SET contact=(
  xmlparse(document
    '<Client>
      <address>
        <street>5401 Julio ave.</street>
        <city>San Jose</city>
        <state>CA</state>
        <zip>95116</zip>
      </address>
      <phone>
        <work>4084633000</work>
        <home>4081111111</home>
        <cell>4082222222</cell>
      </phone>
      <fax>4087776666</fax>
      <email>newemail@someplace.com</email>
    </Client>')
  )
WHERE id = 3227
```

#### Listing 2.25 - An example of an SQL UPDATE

For the second way, you can perform sub-document updates using the TRANSFORM expression, which is a lot more efficient. This allows you to replace, insert, delete or rename nodes in an XML document. You can also change the value of a node without replacing the node itself, typically to change an element or attribute value—which is a very common type of update. This support was added in DB2 9.5.

The TRANSFORM expression is part of the XQuery language, you can use it anywhere you normally use XQuery, for example in a FLWOR expression or in the XMLQUERY function in an SQL/XML statement. The most typical use is in an SQL UPDATE statement to modify an XML document in an XML column.

Listing 2.26 shows the syntax of the TRANSFORM expression.

```
>>-transform--| copy clause |--| modify clause |--| return clause |--<
```

copy clause

```

      .-,-----
      v                                     |
|--copy---$VariableName--:=--CopySourceExpression-+-----|

```

modify clause

```

|--modify--ModifyExpression-----|

```

return clause

```

|--return--ReturnExpression-----|

```

### Listing 2.26 - The syntax of the TRANSFORM expression

The `copy` clause is used to assign to a variable the XML documents you want to process.

In the `modify` clause, you can invoke an `insert`, `delete`, `rename`, or `replace` expression. These expressions allow you to perform updates to your XML document.

For example:

- If you want to add new nodes to the document, you would use the `insert` expression
- To delete nodes from an XML document, use the `delete` expression
- To rename an element or attribute in the XML document, use the `rename` expression
- To replace an existing node with a new node or sequence of nodes, use the `replace` expression. The `replace` value of the expression can only be used to change the value of an element or attribute.

The `return` clause returns the result of the transform expression.

*Listing 2.27* shows an example of an `UPDATE` statement using the `TRANSFORM` expression.

```

(1)-- UPDATE customers
(2)-- SET contactinfo = xmlquery( 'declare default element namespace
(3)--                               "http://posample.org";
(4)--   transform
(5)--   copy $newinfo := $c
(6)--       modify do insert <email2>my2email.gm.com</email2>
(7)--           as last into $newinfo/customerinfo
(8)--   return $newinfo' passing contactinfo as "c")

```

```
(9)-- WHERE id = 100
```

### Listing 2.27 - An UPDATE using the TRANSFORM expression

In the above example, lines (1), (2), and (9) are part of the SQL UPDATE syntax. In Line (2) the XMLQUERY function is invoked, which calls the transform expression in line (4). The transform expression block goes from line (4) to line (8), and it is used to insert a new node into the XML document containing the *email* element. Note that updating the elements in an XML document through a view is not supported.

Deleting entire XML documents from tables is as straightforward as when using the SELECT statement in SQL/XML. Use the SQL DELETE statement and specify any necessary WHERE predicates.

### 2.3.9 XML indexing

In an XML document, indexes can be created for elements, attributes, or for values (text nodes). Below are some examples. Assume the table below was created:

```
CREATE TABLE customer(info XML)
```

And assume the XML document in Listing 2.28 is one of the documents stored in the table.

```
<customerinfo Cid="1004">
  <name>Matt Foreman</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Toronto</city>
    <state>Ontario</state>
    <pcode>M3Z-5H9</pcode>
  </addr>
  <phone type="work">905-555-4789</phone>
  <phone type="home">416-555-3376</phone>
  <assistant>
    <name>Peter Smith</name>
    <phone type="home">416-555-3426</phone>
  </assistant>
</customerinfo>
```

### Listing 2.28 - The XML document to use in the examples related to XML indexes

The statement shown in *Listing 2.29* creates an index on the attribute *cid*

```
CREATE UNIQUE INDEX idx1 ON customer(info)
  GENERATE KEY USING
  xmlpattern '/customerinfo/@Cid'
  AS sql DOUBLE
```

### Listing 2.29 - An index on attribute Cid

The statement shown in *Listing 2.30* creates an index on the element *name*

```
CREATE INDEX idx2 ON customer(info)
  GENERATE KEY USING
  xmlpattern '/customerinfo/name'
  AS sql VARCHAR(40)
```

### **Listing 2.30 - An index on element name**

The statement in *Listing 2.31* creates an index on all elements *name*

```
CREATE INDEX idx3 ON customer(info)
  GENERATE KEY USING
  xmlpattern '//name'
  AS sql VARCHAR(40);
```

### **Listing 2.31 - An index on all elements name**

The statement in *Listing 2.32* creates an index on all text nodes (all values). This is not recommended, as it would be too expensive to maintain the index for update, delete and insert operations, and the index would be too large.

```
CREATE INDEX idx4 ON customer(info)
  GENERATE KEY USING
  xmlpattern '//text()'
  AS sql VARCHAR(40);
```

### **Listing 2.32 - An index on all text nodes (Not recommended)**

## **2.4 Working with XML Schemas**

DB2 allows you to insert an XML document into the database if it is well-formed. If it's not, you will receive an error at insertion time. On the other hand, DB2 does not force you to validate a XML document. If you wish to have an XML document validated, you have several alternatives as we will discuss in this section.

### **2.4.1 Registering your XML Schemas**

XML Schemas are stored in the DB2 databases in what is called an XML Schema repository. To add an XML Schema to a repository, you use the REGISTER XMLSCHEMA command.

For example, let's say you have an XML document stored in file `order.xml` as shown in *Figure 2.10*

```
<?xml version="1.0" encoding="UTF-8"?>
  <po:PurchaseOrder xmlns:po="http://www.test.com/po">
    <Header>
      <Id>1</Id>
      <date>2004-01-29</date>
      <description>purchase order</description>
      <value>20</value>
      <status>shipped</status>
    </Header>
    <Items>
      <Item>
        <ItemDescription color="red" weight="5">
          <Name>Widget C</Name>
          <SKU>1</SKU>
          <Price>30</Price>
          <Comment>no comment</Comment>
        </ItemDescription>
        <NumberOrdered>1</NumberOrdered>
      </Item>
    </Items>
    <Customer type="regualar">
      <Name>Manoj K Sardana</Name>
      <Address>ring road, bangalore</Address>
      <Phone>918051055109</Phone>
      <email>msardana@in.ibm.com</email>
    </Customer>
  </po:PurchaseOrder>
```

**Figure 2.10 - The order.xml file containing an XML document**

Now, let's say you have an XML Schema document stored in file `order.xsd` as shown in *Figure 2.11*

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.test.com/po"
  xmlns:po="http://www.test.com/po"
  xmlns:head="http://www.test.com/header"
  xmlns:prod = "http://www.test.com/product"
  xmlns:cust = "http://www.test.com/customer">

  <xsd:import namespace="http://www.test.com/product" schemaLocation="product.xsd" />
  <xsd:import namespace="http://www.test.com/customer" schemaLocation="customer.xsd" />
  <xsd:import namespace="http://www.test.com/header" schemaLocation="header.xsd" />
  <xsd:complexType name="itemType">
    <xsd:sequence>
      <xsd:element name="Item" minOccurs="1" maxOccurs="unbounded" >
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="ItemDescription" type="prod:prodType" />
            <xsd:element name="NumberOrdered" type="xsd:integer" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="potype">
    <xsd:sequence>
      <xsd:element name="Header" type="head:headerType" />
      <xsd:element name="Items" type="po:itemType" />
      <xsd:element name="Customer" type="cust:customerType" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="PurchaseOrder" type="po:potype" />

</xsd:schema>

```

**Figure 2.11 - The order.xsd file containing an XML schema**

In this XML Schema document we highlight with an ellipse the following:

- `<xsd:schema ...>`: Indicates it's a XML Schema document
- `<xsd:import ...>`: We import other xsd files (other XML Schemas) that would be part of this bigger XML Schema.
- `minOccurs="1"`: An example of an XML Schema "rule", where for element `Item` we say that it should occur at least one time, or in other words, there should be at least one `Item` element.

Next, in order to register the XML Schema to the database, a script similar to the one shown in *Listing 2.33* below could be used. The script includes comments that make it self-explanatory.

```

-- CONNECT TO THE DATABASE
CONNECT TO SAMPLE;

-- REGISTER THE MAIN XML SCHEMA
REGISTER XMLSCHEMA http://www.test.com/order FROM D:\example3\order.xsd AS

```

```
order;

-- ADD XML SCHEMA DOCUMENT TO MAIN SCHEMA
ADD XMLSCHEMA DOCUMENT TO order ADD http://www.test.com/header FROM
D:\example3\header.xsd;

-- ADD XML SCHEMA DOCUMENT TO MAIN SCHEMA
ADD XMLSCHEMA DOCUMENT TO order ADD http://www.test.com/product FROM
D:\example3\product.xsd;

-- ADD XML SCHEMA DOCUMENT TO MAIN SCHEMA
ADD XMLSCHEMA DOCUMENT TO order ADD http://www.test.com/customer FROM
D:\example3\customer.xsd;

-- COMPLETE THE SCHEMA REGISTRATION
COMPLETE XMLSCHEMA order;
```

**Listing 2.33 - A sample script showing the steps to register an XML schema**

To review this information later you can SELECT the information from the Catalog tables as shown in *Listing 2.34* below.

```
SELECT CAST(OBJECTSCHEMA AS VARCHAR(15)), CAST(OBJECTNAME AS VARCHAR(15))
FROM syscat.xsobjects
WHERE OBJECTNAME='ORDER';
```

**Listing 2.34 - Retrieving XML schema information from the DB2 Catalog tables**

### 2.4.2 XML Schema validation

Once your XML Schemas have been registered in DB2, you can validate your XML documents in two ways:

- Use the XMLVALIDATE function during an INSERT
- Use a BEFORE Trigger

*Figure 2.12* shows an example where the XML document shown in *Figure 2.10* is validated according to the XML Schema shown in *Figure 2.11*.



```

DROP TABLE t1;
CREATE TABLE t1 (po xml);

INSERT INTO t1 VALUES(xmlvalidate(xmlparse(document('<?xml version="1.0" encoding="UTF-8"?>
<po:PurchaseOrder xmlns:po="http://www.test.com/po">
  <Header>
    <Id>1</Id>
    <date>2004-01-29</date>
    <description>purchase order</description>
    <value>20</value>
    <status>shipped</status>
  </Header>
  <Items>
    <Item>
      <ItemDescription color="red" weight="5">
        <Name>Widget C</Name>
        <SKU>1</SKU>
        <Price>30</Price>
        <Comment>no comment</Comment>
      </ItemDescription>
      <NumberOrdered>1</NumberOrdered>
    </Item>
  </Items>
  <Customer type="regular">
    <Name>Manoj K Sardana</Name>
    <Address>ring road, bangalore</Address>
    <Phone>918051055109</Phone>
    <email>msardana@in.ibm.com</email>
  </Customer>
</po:PurchaseOrder>')) ACCORDING TO XMLSCHEMA ID order));

```

### Figure 2.12 - XML Schema validation using XMLVALIDATE

To test if an XML document has been validated, you can use the “IS VALIDATED” predicate on a CHECK constraint.

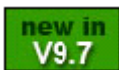
You can validate XML documents in a column using different XML schemas. This is important for easy migration from version 1 to version 2 of an XML schema. In the same XML column, you may also find XML documents with no validation at all. This is useful if documents are received from trusted and non-trusted sources where only the later require schema validation.

### 2.4.3 Other XML support

Small XML documents can now be in-lined with the base table. This means that the XML data is stored in the same place as the relational data, and can take advantage of the same compression mechanisms as regular relational data. Larger XML documents are stored in a separate internal object, which can also be compressed.

DB2 software also supports XML Schema evolution. This means that if your XML Schema changes, you can update the XML Schema easily with the UPDATE XMLSCHEMA command. If the changes to the XML Schema are too drastic, you are likely to get some errors.

In DB2 XML decomposition or “shredding” is also supported. This is the “old” method to store XML in databases, and is what other vendors use to store XML. DB2 still supports this method if you wish to use it; but we recommend pureXML. DB2 also supports the XML Extender, also using the old method to store XML, but this extender will no longer be enhanced.



With DB2 9.7 all the benefits of pureXML has been extended to database partitions commonly used for data warehouses. Database Partitioning Feature (DPF) is offered with DB2 Enterprise Edition.

## 2.5 Exercises

Throughout this chapter, you have seen several examples of SQL/XML and XQuery syntax and have been introduced to the DB2 Command Editor and IBM Data Studio. In this exercise, you will test your SQL/XML and XQuery knowledge while gaining experience with these tools. We will use the **mydb** database created using the **table\_creation.txt** script file which was explained earlier in this chapter (*Listing 2.5*).

### Procedure

1. Create the **mydb** database and load the XML data, as discussed earlier in the chapter. The file `table_creation.txt` is included in the accompanying file **Exercise\_Files\_DB2\_Application\_Development.zip** under the Chapter 2 folder. Run the `table_creation.txt` script file from a DB2 Command Window or Linux shell as follows:

```
db2 -tvf table_creation.txt
```

2. If the script fails in any of the steps, try to figure out the problem by reviewing the error messages. A typical problem when running the script is that you may need to change the paths of the files as they may be located in different directories. You can always drop the database and start again issuing this command from the DB2 Command Window or Linux shell:

```
db2 drop database mydb
```

3. If you receive an error while trying to drop the database because of active connections, issue this command first:

```
db2 force applications all
```

4. After successfully running the script, use the DB2 Control Center, or IBM Data Studio to verify that the **items** and **clients** tables are created and that they contain 4 and 7 rows respectively.
5. With the **mydb** database created and with the two tables loaded, you can now connect to it, and perform the queries shown in *Listings 2.7* through *2.19*

## 2.6 Summary

This chapter introduced you to XML and pureXML technology. XML document usage is growing exponentially due to Web 2.0 tools and techniques as well as SOA. By storing XML documents in a DB2 database you can take advantage of security, performance, and coding flexibility using pureXML. pureXML is a technology that allows you to store the XML documents in parsed-hierarchical format, as a tree, and this is done at database insertion time. At query time, there is no need to parse the XML document in order to build a tree before processing. The tree for the XML document was already built and stored in the database. In addition, pureXML technology uses a native XML engine that understands XQuery; therefore, there is no need to map XQuery to SQL which is what is done in other RDBMS products.

The chapter also talked about how to insert, delete, update and query XML documents using SQL/XML and XQuery. It also discussed XML indexes, XML Schema, and other features such as compression and XML Schema evolution.

## 2.7 Review questions

1. Why is it a good idea to store XML documents in a database as opposed to files?
2. What are the two types of databases for storing XML data?
3. What are the two main characteristics of pureXML?
4. Why would using pureXML be better for application performance?
5. How can you insert an XML document into a DB2 database?
6. Which of the following can be used to retrieve XML data in DB2?
  - A. SQL
  - B. SQL/XML
  - C. XQuery
  - D. B and C
  - E. All of the above
7. Which of the following is not a SQL/XML function?
  - A. XMLQUERY

- B. XMLTABLE
  - C. XMLCAST
  - D. XMLVALIDATE
  - E. XMLNAVIGATE
8. Which of the following is not an XQuery function
- A. db2-fn:xmlcolumn
  - B. db2-fn:sqlquery
  - C. XMLQUERY
  - D. All of the above
  - E. None of the above
9. How can you update an XML document?
- A. Use the SQL INSERT statement
  - B. Use the TREEUPDATE expression
  - C. Use the TRANSFORM expression
  - D. All of the above
  - E. None of the above
10. Which of the following statements is true?
- A. XML Indexes can be created for elements, and attributes, but not values
  - B. XML Schemas are stored in the XML Schema repository, a separate internal file not related to the database.
  - C. You can validate an XML document with an AFTER trigger
  - D. All of the above
  - E. None of the above

# 3

## Chapter 3 – Stored procedures, UDFs, triggers, and data Web services

This chapter focuses on data server-side development using stored procedures, user-defined functions (UDFs) and triggers.

A stored procedure is a database application object that can encapsulate SQL statements and business logic. Keeping part of the application logic in the database provides performance improvements as the amount of network traffic between the application and the database is considerably reduced. In addition, stored procedures provide a centralized location to store your code, so other applications can reuse the same stored procedures.

A UDF allows you to extend the SQL language and encapsulate your business logic.

Triggers are database objects that allow database administrators to have the data server automatically verify the validity of data before insertion, or to audit data after it has been modified.

Data Web services externalize information from a DB2 database by converting SQL scripts or stored procedures into a Web service. This provides a flexible, inexpensive, and convenient solution to companies that need to share information stored in their databases.

In this chapter you will learn about:

- The basics of SQL PL and Java stored procedures
- Developing User-defined functions
- Creating Data Web Services
- Working with IBM Data Studio
- Creating BEFORE, AFTER and INSTEAD OF triggers.

### 3.1 Stored procedures: The big picture

DB2 stored procedures can be written using SQL PL (SQL Procedural Language), C/C++, Java, COBOL, CLR (Common Language Runtime) supported languages, and OLE. In this chapter, we focus on SQL PL procedures because of their popularity, good performance

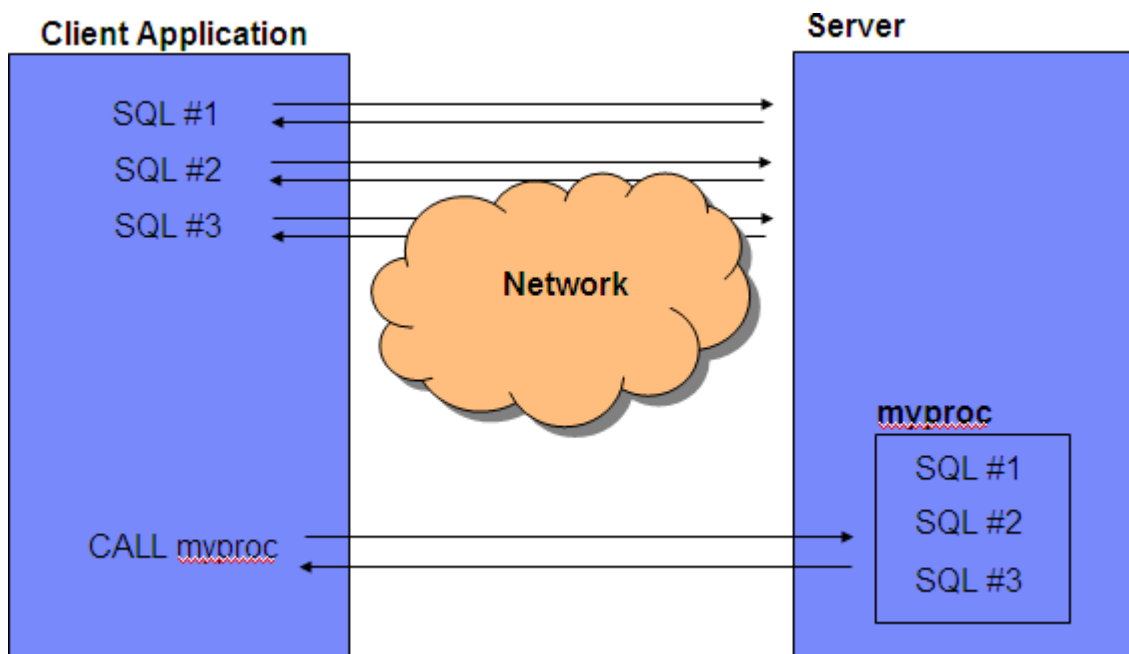
new in  
V9.7

and simplicity. The SQL PL language is based on the SQL/PSM (SQL Persisted Stored Modules) standard. We also discuss briefly how to create a Stored Procedure using the Java language.

**Note:**

New with DB2 9.7 is the support for PL/SQL (Procedural Language / SQL) which is Oracle's data server proprietary procedural language extension to SQL/PSM and is used with stored procedures. The DB2 Express-C edition does not currently support PL/SQL.

Figure 3.1 illustrates how stored procedures work.



**Figure 3.1 – Network traffic reduction with stored procedures**

At the top left corner of the figure, you see several SQL statements executed one after the other. Each SQL is sent from the client to the data server, and the data server returns the result back to the client. If many SQL statements are executed like this, network traffic increases. On the other hand, at the bottom, you see an alternate method that incurs less network traffic. This second method calls a stored procedure *myproc* stored on the server, which contains the same SQL; and then at the client (on the left side), the CALL statement is used to call the stored procedure. This second method is more efficient, as there is only one call statement that goes through the network, and one result set returned to the client.

Stored procedures can also be helpful for security purposes in your database. For example, you can let users access tables or views only through stored procedures; this

helps lock down the server and keep users from accessing information they are not supposed to access. This is possible because users do not require explicit privileges on the tables or views they access through stored procedures; they just need to be granted sufficient privilege to invoke the stored procedures.

**Note:**

For more information about SQL PL stored procedures, watch this video:

<http://www.channeldb2.com/video/video/show?id=807741:Video:4343>

### 3.2 Working with IBM Data Studio

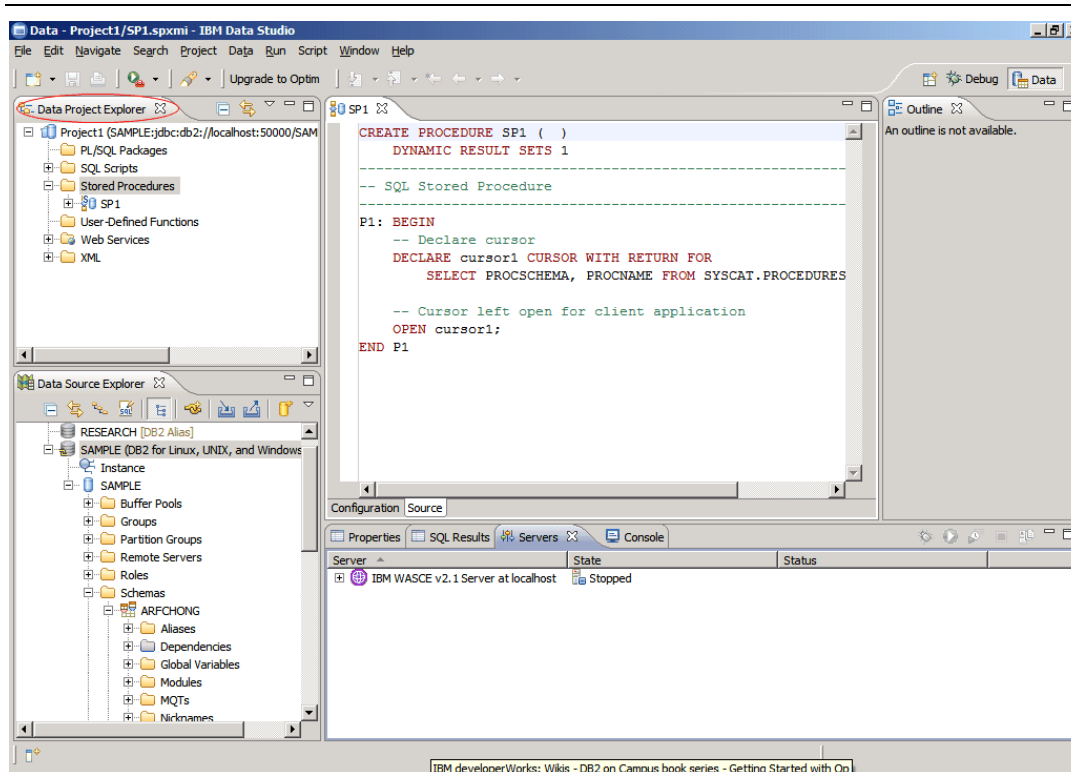
IBM Data Studio 2.2 will be used in this chapter to develop stored procedures, UDFs, and data Web services. IBM Data Studio 2.2 is free. It is not included with DB2, but provided as a separate image; and it comes in two flavors:

- IDE: Allows you to share the same Eclipse (shell sharing) with other products such as InfoSphere Data Architect and Rational products. It also provides support for Data Web services.
- Stand-alone: This version provides almost the same functionality as the IDE version but without support for Data Web services and without shell sharing. The footprint for this version is a lot smaller.

**Note:**

For a thorough coverage about IBM Data Studio, refer to the free eBook [Getting started with IBM Data Studio for DB2](#) which is part of this DB2 on Campus free book series.

In this chapter we use the IDE version because we demonstrate how to work with Data Web services. The Data Studio images can be downloaded from [www.ibm.com/db2/express](http://www.ibm.com/db2/express). *Figure 3.2* shows IBM Data Studio 2.2.



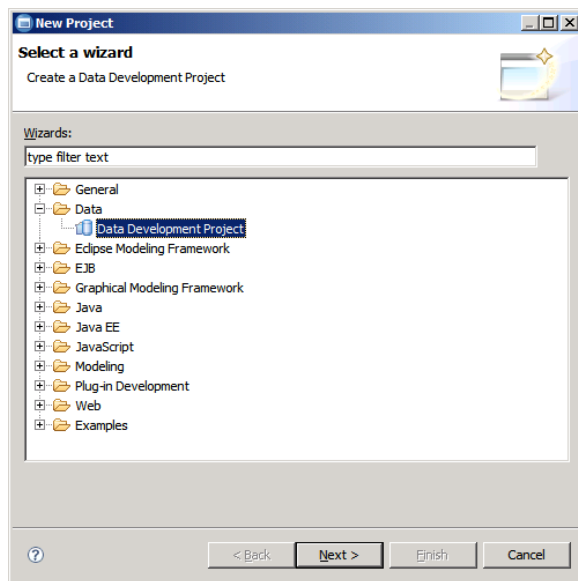
**Figure 3.2 – IBM Data Studio 2.2**

In this chapter we focus on the *Data Project Explorer* view highlighted at the top right corner of the figure. This view focuses on data server-side development.

### 3.2.1 Creating a project

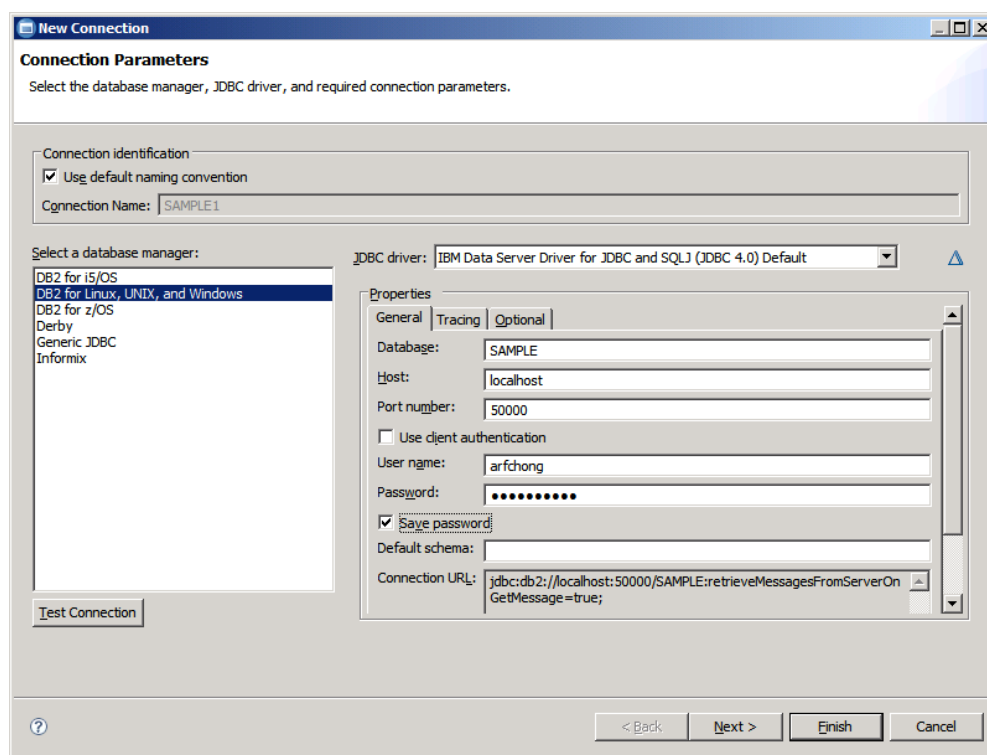
Before you can develop stored procedures, UDFs or Data Web services in Data Studio, you need to create a project. From the Data Studio menu, choose *File -> New -> Project* and choose *Data Development Project*. This is shown in *Figure 3.3*.





**Figure 3.3 – Creating a data development project**

Follow the steps from the wizard to input a name for your project, and indicate which database you want your project associated with. If you do not have any existing database connection, click on the *New* button in the *Select Connection* panel, and a window as shown in *Figure 3.4* will appear.



**Figure 3.4 – New connection parameters**

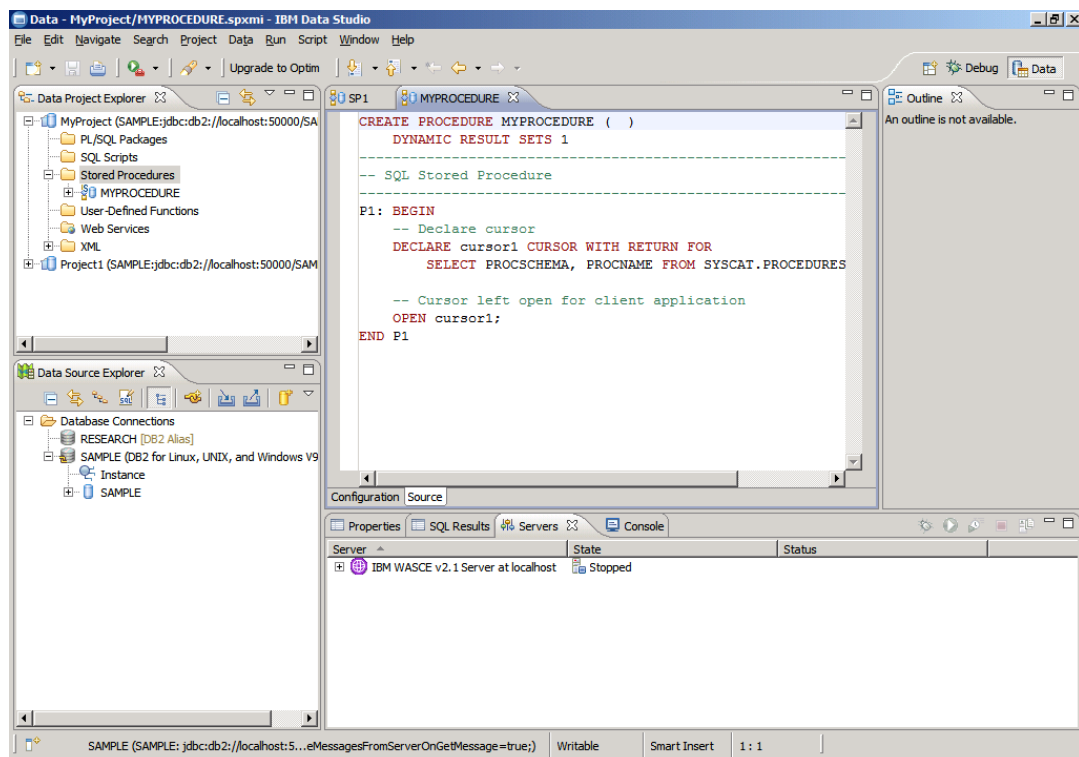
In *Figure 3.4*, make sure to choose *DB2 for Linux, UNIX and Windows* in the *Select a database manager* field on the left side of the figure. For the *JDBC driver* drop down menu, the default after choosing *DB2 for Linux, UNIX and Windows* is the JDBC type 4 driver listed as *IBM Data Server Driver for JDBC and SQLJ (JDBC 4.0) Default*. Use this default driver and complete the specified fields. For the *host* field, you can input an IP address or a hostname. In the example IBM Data Studio and the DB2 database reside on the same computer, so *localhost* was chosen. Ensure to test that your connection to the database is working by clicking on the *Test Connection* button shown on the lower left corner of the figure. If the connection test was successful, click *Finish* and the database name will be added to the list of connections you can associate your project to. Select the database, then click *Finish* and your project should be displayed on the *Data Project Explorer* view. In this view, if you click on the "+" symbol, you can drill down the project to see different folders such as PL/SQL packages, SQL scripts, stored procedures, etc.

### 3.2.2 Creating a stored procedure

To create a Java, PL/SQL or SQL PL stored procedure in Data Studio, follow the steps below. Note that stored procedures in other languages cannot be created from Data Studio. In the following steps, we choose SQL (representing SQL PL) as the language for the stored procedure, however similar steps apply to Java and PL/SQL languages.

### Step 1: Write or generate the stored procedure code

When you want to create a stored procedure, right-click on the Stored Procedures folder and choose *New -> Stored Procedure*. Complete the information requested in the *New Stored Procedure* wizard such as the project to associate the procedure with, the name and language of the procedure, and the SQL statements to use in the procedure. By default, Data Studio gives you an example SQL statement. Take all the defaults for all the other panels, or at this point, you can click *Finish* and a stored procedure is created using some template code and the SQL statement provided before as an example. This is shown in *Figure 3.5*.

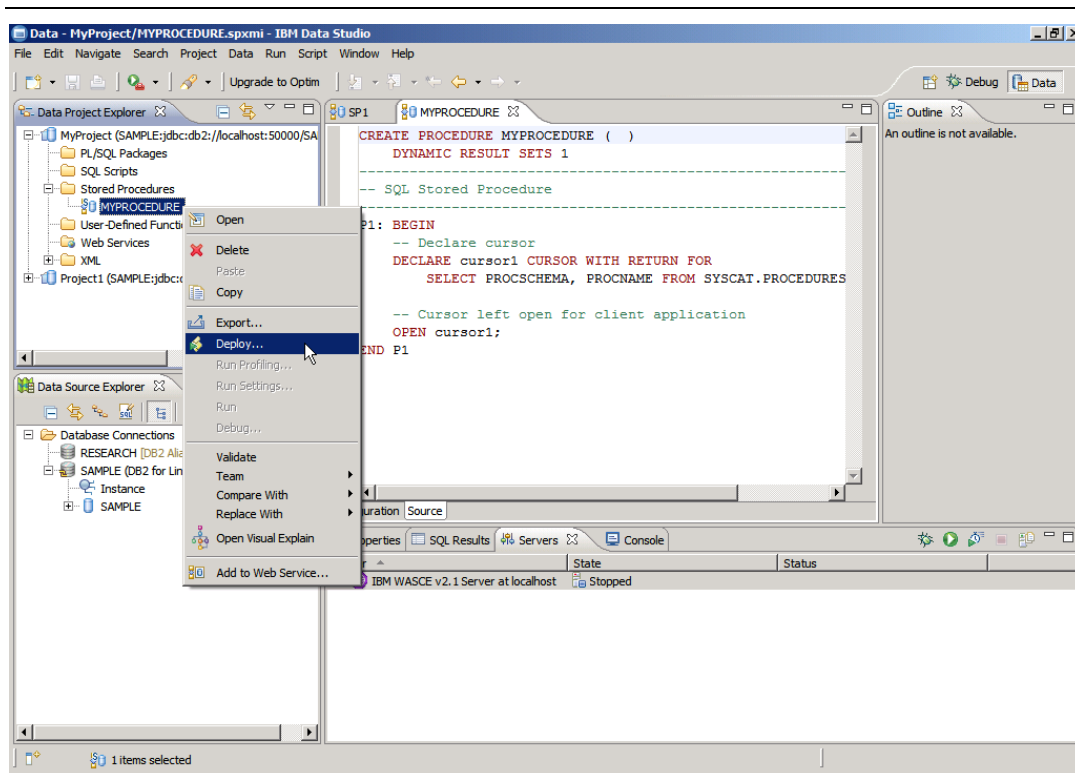


**Figure 3.5 – A sample stored procedure**

In *Figure 3.5*, the code for the sample stored procedure *MYPROCEDURE* was generated. You can replace all of this code with your own code. For simplicity, we will continue in this book using the above sample stored procedure as if we had written it.

### Step 2: Deploy a stored procedure

Once the stored procedure is created, to deploy it, select it from the *Data Project Explorer* view, right-click on it, and then choose *Deploy*. Deploying a stored procedure is basically executing the **CREATE PROCEDURE** statement, compiling the procedure and storing it in the database. *Figure 3.6* illustrates this step.

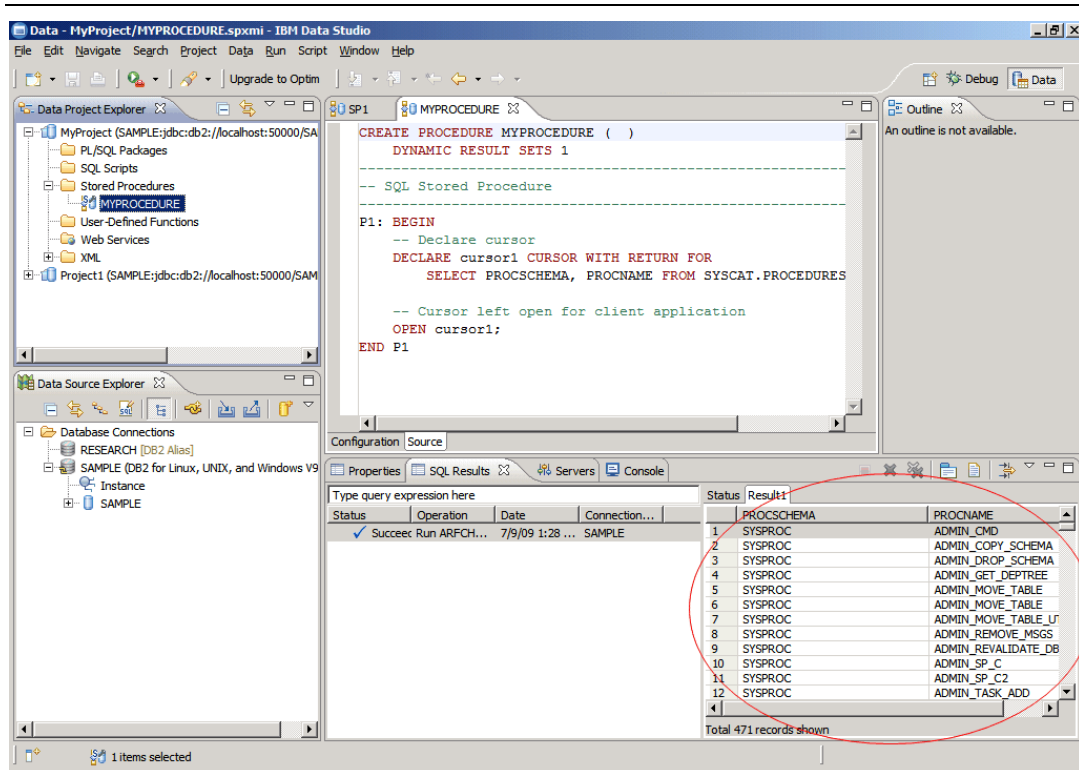


**Figure 3.6 – Deploying a stored procedure**

After clicking *Deploy*, in the *Deploy options* panel, taking the defaults and clicking on *Finish* is normally good enough.

**Step 4: Run a stored procedure**

Once the stored procedure has been deployed, you can run it by right-clicking on it and choosing *Run*. The results would appear in the *Results* tab at the bottom right corner of the Data Studio workbench window as shown in *Figure 3.7*.



**Figure 3.7 – Output after running a stored procedure**

To run a stored procedure from the DB2 Command Window or the Command Editor, you can use the `CALL <procedure name>` statement. Remember you first need to connect to the database since this is where the stored procedure resides. *Figure 3.8* illustrates this.

```

DB2 CLP - DB2COPY1
C:\Program Files\IBM\SQLLIB\BIN>db2 connect to sample

Database Connection Information
Database server          = DB2/NT 9.7.0
SQL authorization ID    = ARFCHONG
Local database alias    = SAMPLE

C:\Program Files\IBM\SQLLIB\BIN>db2 call myprocedure

Result set 1
-----
PROCSCHEMA                                PROCNAME
-----
SYSPROC                                   ADMIN_CMD
SYSPROC                                   ADMIN_COPY_SCHEMA
SYSPROC                                   ADMIN_DROP_SCHEMA

```

**Figure 3.8 – Calling a stored procedure from the DB2 Command Window**

Just like you can call a stored procedure from the DB2 Command Window, you can also do so from a Java program, a C program, a Visual Basic program, and so on. You just need to use the correct syntax for the given language.

### 3.3 SQL PL stored procedures basics

SQL PL stored procedures are easy to create and learn. They have the best performance in DB2. SQL PL stored procedures (or simply “SQL stored procedures”) are the focus of this chapter.

#### 3.3.1 Stored procedure structure

The basic store procedure syntax is shown below.

```

CREATE PROCEDURE proc_name [( {optional parameters} )]
    [optional procedure attributes]
    <statement>

```

Where <statement> is a single statement, or a set of statements grouped by BEGIN [ATOMIC] ... END

#### 3.3.2 Optional stored procedure attributes

The following describes some of the optional stored procedure attributes:

- **LANGUAGE SQL**  
This attribute indicates the language the stored procedure will use. LANGUAGE SQL is the default value. For other languages, such as Java or C use LANGUAGE JAVA or LANGUAGE C, respectively.
- **RESULT SETS <n>**  
This is required if your stored procedure will be returning n result sets.
- **SPECIFIC my\_unique\_name**  
This is a unique name that can be given to a procedure. A stored procedure can be overloaded, that is, several stored procedures can have the same name, but with different number of parameters. By using the SPECIFIC keyword you can provide one unique name for each of these stored procedures, and this can ease stored procedure management. For example, to drop a stored procedure using the SPECIFIC keyword, you can issue this statement: **DROP SPECIFIC PROCEDURE**. If the SPECIFIC keyword had not been used you would have had to use a **DROP PROCEDURE** statement and put the name of the procedure with all the parameters so DB2 would know which of the overloaded procedures you wanted to drop.

### 3.3.3 Parameters

There are three types of parameters in an SQL PL stored procedure:

- **IN** - Input parameter
- **OUT** - Output parameter
- **INOUT** - Input and Output parameter

For example:

```
CREATE PROCEDURE proc(IN p1 INT, OUT p2 INT, INOUT p3 INT)
```

To call the procedure use the CALL statement. For example, to call the above stored procedure you could specify:

```
CALL proc (10,?,4)
```

The question mark (?) is used for OUT parameters in the CALL statement. *Listing 3.1* provides another example of a stored procedure with parameters that you can try.

```
CREATE PROCEDURE P2 ( IN    v_p1 INT,
                    INOUT v_p2 INT,
                    OUT   v_p3 INT)

LANGUAGE SQL
SPECIFIC myP2
BEGIN
    -- my second SQL procedure
    SET v_p2 = v_p2 + v_p1;
    SET v_p3 = v_p1;
END
```

**Listing 3.1 - A stored procedure with parameters**

To call the procedure from the Command Editor you can use:

```
call P2 (3, 4, ?)
```

**3.3.4 Comments in an SQL PL stored procedure**

There are two ways to specify comments in an SQL PL stored procedure:

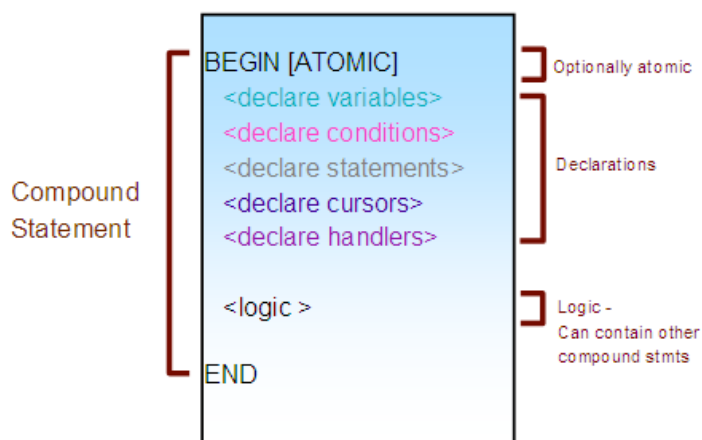
- Using two dashes. For example:
 

```
-- This is an SQL-style comment
```
- Using a format similar to the C language. For example:
 

```
/* This is a C-style coment */
```

**3.3.5 Compound statements**

A compound statement in a stored procedure is a statement consisting of several procedural instructions and SQL statements encapsulated by the keywords BEGIN and END. When the ATOMIC keyword follows the BEGIN keyword, the compound statement is treated as one unit, that is, all of the instructions or statements in the compound statement must be successful in order for the entire compound statement to be successful. If one of the statements is not, then everything is rolled back. *Figure 3.9* shows a compound statement structure.



**Figure 3.9 – Compound statements**

**3.3.6 Variable declaration**

To declare a variable, use the DECLARE statement as shown below.

```
DECLARE var_name <data type> [DEFAULT value];
```

*Listing 3.2* provides some examples.



```
DECLARE temp1 SMALLINT DEFAULT 0;
DECLARE temp2 INTEGER DEFAULT 10;
DECLARE temp3 DECIMAL(10,2) DEFAULT 100.10;
DECLARE temp4 REAL DEFAULT 10.1;
DECLARE temp5 DOUBLE DEFAULT 10000.1001;
DECLARE temp6 BIGINT DEFAULT 10000;
DECLARE temp7 CHAR(10) DEFAULT 'yes';
DECLARE temp8 VARCHAR(10) DEFAULT 'hello';
DECLARE temp9 DATE DEFAULT '1998-12-25';
DECLARE temp10 TIME DEFAULT '1:50 PM';
DECLARE temp11 TIMESTAMP DEFAULT '2001-01-05-12.00.00';
DECLARE temp12 CLOB(2G);
DECLARE temp13 BLOB(2G);
```

**Listing 3.2 - Variable declaration examples****3.3.7 Assignment statements**

To assign a value to a variable, use the SET statement. For example:

```
SET total = 100;
```

The above statement is equivalent to

```
VALUES(100) INTO total;
```

Additionally, any variable can be set to NULL. For example:

```
SET total = NULL;
```

You can assign a value to a variable based on the output of a SELECT statement. For example:

```
SET total = (select sum(c1) from T1);
SET first_val = (select c1 from T1 fetch first 1 row only)
```

An error condition is raised if more than one value is fetched from a table and you are trying to assign it to a single variable. If you need to store more information than a single value, use arrays, or cursors.

You can also set variables according to external database properties or special DB2 register variables. For example:

```
SET sch = CURRENT SCHEMA;
```

### 3.3.8 Cursors

A cursor is a result set holding the result of a SELECT statement. The syntax to declare, open, fetch, and close a cursor is shown in *Listing 3.3*.

```
DECLARE <cursor name> CURSOR [WITH RETURN <return target>]
    <SELECT statement>;
OPEN <cursor name>;
FETCH <cursor name> INTO <variables>;
CLOSE <cursor name>;
```

#### Listing 3.3 - Syntax to work with cursors

When a cursor is declared, the WITH RETURN clause can be used with these values:

- CLIENT: the result set will return to client application
- CALLER: the result set is returned to client or stored procedure that made the call

*Listing 3.4* below provides an example of a stored procedure using a cursor.

```
CREATE PROCEDURE set()
DYNAMIC RESULT SETS 1
LANGUAGE SQL
BEGIN
DECLARE cur CURSOR WITH RETURN TO CLIENT
    FOR SELECT name, dept, job
        FROM staff
        WHERE salary > 20000;
OPEN cur;
END
```

#### Listing 3.4 - A stored procedure using a cursor

### 3.3.9 Flow control

Like in many other languages, SQL PL has several statements that can be used to control the flow of the logic. Below we list some of the flow control statements supported:

- CASE (selects an execution path, simple search)
- IF
- FOR (executes body for each row of table)
- WHILE
- ITERATE (forces next iteration. Similar to CONTINUE in C)
- LEAVE (leaves a block or loop. "Structured Goto")
- LOOP (infinite loop)
- REPEAT

- GOTO
- RETURN
- CALL (procedure call)

### 3.3.10 Errors and condition handlers

In DB2, the SQLCODE and SQLSTATE keywords are used to determine the successful or unsuccessful execution of an SQL statement. These keywords need to be explicitly declared in the outermost scope of the procedure as follows:

```
DECLARE SQLSTATE CHAR(5);  
DECLARE SQLCODE INT;
```

DB2 will set the values of the above keywords automatically after each SQL operation. For the SQLCODE, the values are set as follows:

- = 0, successful.
- > 0, successful with warning
- < 0, unsuccessful
- = 100, no data was found. (i.e.: FETCH statement returned no data)

For the SQLSTATE, the values are set as follows:

- success: SQLSTATE '00000'
- not found: SQLSTATE '02000'
- warning: SQLSTATE '01XXX'
- exception: all other values

The SQLCODE is RDBMS specific, and more detailed than the SQLSTATE. The SQLSTATE is standard among RDBMSs but is very general in nature. Several SQLCODEs may match one SQLSTATE.

A condition can be raised by any SQL statement and would match an SQLSTATE. For example, a specific condition like SQLSTATE '01004' is raised when a value is truncated during an SQL operation. Rather than using SQLSTATE '01004' to test for this condition, names can be assigned. In this particular example, the name **trunc** can be assigned to condition SQLSTATE '01004' as shown below.

```
DECLARE trunc CONDITION FOR SQLSTATE '01004'
```

Other predefined general conditions are:

- SQLWARNING

- SQLEXCEPTION
- NOT FOUND

### 3.3.10.1 Condition handling

To handle a condition, you can create a condition handler which must specify:

- Which conditions it handles
- Where to resume execution (based on the type of the handler: CONTINUE, EXIT or UNDO)
- Which actions to perform to handle the condition. These actions can be any statement, including control structures.

If an SQLEXCEPTION condition is raised, and there is no handler, the procedure terminates and returns to the client with an error.

### 3.3.10.2 Types of handlers

There are three types of handlers:

- **CONTINUE** – This handler is used to indicate that after an exception is raised, and the handler handles the condition, the flow will CONTINUE to the next statement after the statement that raised the condition.
- **EXIT** – This handler is used to indicate that, after an exception is raised, and the handler handles the condition, the flow will go to the end of the procedure.
- **UNDO** – This handler is used to indicate that after an exception is raised, and the handler handles the condition, the flow will go to the end of the procedure, and will undo or roll back any statements performed.

Figure 3.10 illustrates the different condition handlers and their behavior.

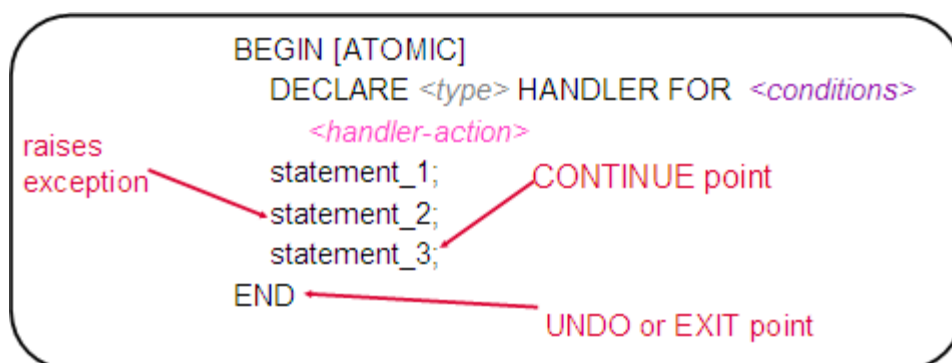


Figure 3.10 – Type of condition handlers

### 3.3.11 Calling stored procedures

The following code snippets illustrated in *Listings 3.5, 3.6 and 3.7* show how to CALL stored procedures from a CLI/ODBC, VB.NET, and Java program respectively.

```
SQLCHAR *stmt = (SQLCHAR *)
"CALL MEDIAN_RESULT_SET( ? )" ;
SQLDOUBLE   sal = 20000.0; /* Bound to parameter marker in stmt */
SQLINTEGER  salind = 0;    /* Indicator variable for sal */

sqlrc = SQLPrepare(hstmt, stmt, SQL_NTS);
sqlrc = SQLBindParameter(hstmt, 1, SQL_PARAM_OUTPUT,
    SQL_C_DOUBLE, SQL_DOUBLE, 0, 0, &sal, 0, &salind);
SQLExecute(hstmt);

if (salind == SQL_NULL_DATA)
    printf("Median Salary = NULL\n");
else
    printf("Median Salary = %.2f\n\n", sal );

/* Get first result set */
sqlrc = StmtResultPrint(hstmt);
/* Check for another result set */
sqlrc = SQLMoreResults(hstmt);
if (sqlrc == SQL_SUCCESS) {
    /* There is another result set */
    sqlrc = StmtResultPrint(hstmt);
}
}
```

#### Listing 3.5 - Example calling a stored procedure from a CLI/ODBC application

For more details, see the DB2 sample file: `sqllib/samples/sqlproc/rsultset.c`

Try

```
` Create a DB2Command to run the stored procedure
Dim procName As String = "TRUNC_DEMO"
Dim cmd As DB2Command = conn.CreateCommand()
Dim parm As DB2Parameter

cmd.CommandType = CommandType.StoredProcedure
cmd.CommandText = procName

` Register the output parameters for the DB2Command
parm          = cmd.Parameters.Add("v_lastname", DB2Type.VarChar)
parm.Direction = ParameterDirection.Output
```

## 102 Getting started with DB2 application development

---

```
parm          = cmd.Parameters.Add("v_msg", DB2Type.VarChar)
parm.Direction = ParameterDirection.Output

` Call the stored procedure
Dim reader As DB2DataReader = cmd.ExecuteReader

Catch myException As DB2Exception
    DB2ExceptionHandler(myException)
Catch
    UnhandledExceptionHandler()
End Try
```

### Listing 3.6 - Example calling a stored procedure from a VB.NET application

```
try
{
    // Connect to sample database
    String url = "jdbc:db2:sample";
    con = DriverManager.getConnection(url);

    CallableStatement cs = con.prepareCall("CALL trunc_demo(?, ?)");

    // register the output parameters
    callStmt.registerOutParameter(1, Types.VARCHAR);
    callStmt.registerOutParameter(2, Types.VARCHAR);

    cs.execute();
    con.close();
}
catch (Exception e)
{
    /* exception handling logic goes here */
}
```

### Listing 3.7 - Example calling a stored procedure from a Java application

#### 3.3.12 Dynamic SQL

In dynamic SQL, as opposed to static SQL, the entire SQL statement is not known at run time. For example if *col1* and *tablename* are variables in this statement, then we are dealing with dynamic SQL:

```
'SELECT ' || col1 || ' FROM ' || tablename;
```

Dynamic SQL is recommended for DDL to avoid dependency problems and package invalidation. It is also required to implement recursion.

Dynamic SQL can be executed using two approaches:

- Using the EXECUTE IMMEDIATE statement – This is ideal for single execution SQL
- Using the PREPARE statement along with the EXECUTE statement - This is ideal for multiple execution SQL

The code snippet illustrated in *Listing 3.8* provides an example of Dynamic SQL using the two approaches. The example assumes a table T2 has been created with this definition:

```
CREATE TABLE T2 (c1 INT, c2 INT)

CREATE PROCEDURE dyn1 (IN value1 INT, IN value2 INT)
  SPECIFIC dyn1
  BEGIN
    DECLARE stmt varchar(255);
    DECLARE st STATEMENT;

    SET stmt = 'INSERT INTO T2 VALUES (?, ?)';

    PREPARE st FROM stmt;

    EXECUTE st USING value1, value1;
    EXECUTE st USING value2, value2;

    SET stmt = 'INSERT INTO T2 VALUES (9,9)';
    EXECUTE IMMEDIATE stmt;
  END
```

**Listing 3.8 - An example of a stored procedures using dynamic SQL**

### 3.4 Java Stored Procedures

A Java stored procedure in DB2 is created using the CREATE PROCEDURE statement which provides the definition for the procedure, and points to an external Java application. Java stored procedures can be easily created using IBM Data Studio. You can follow the same steps as discussed in section 3.2, choosing *Java* for the language. IBM Data Studio will also let you choose between JDBC or SQLJ for the procedure. If you take all the defaults when creating the Java stored procedure, a corresponding Java application will be generated for you. *Figure 3.11* illustrates IBM Data Studio with a Java stored procedure created. *Figure 3.12* shows the corresponding Java source code.

## 104 Getting started with DB2 application development

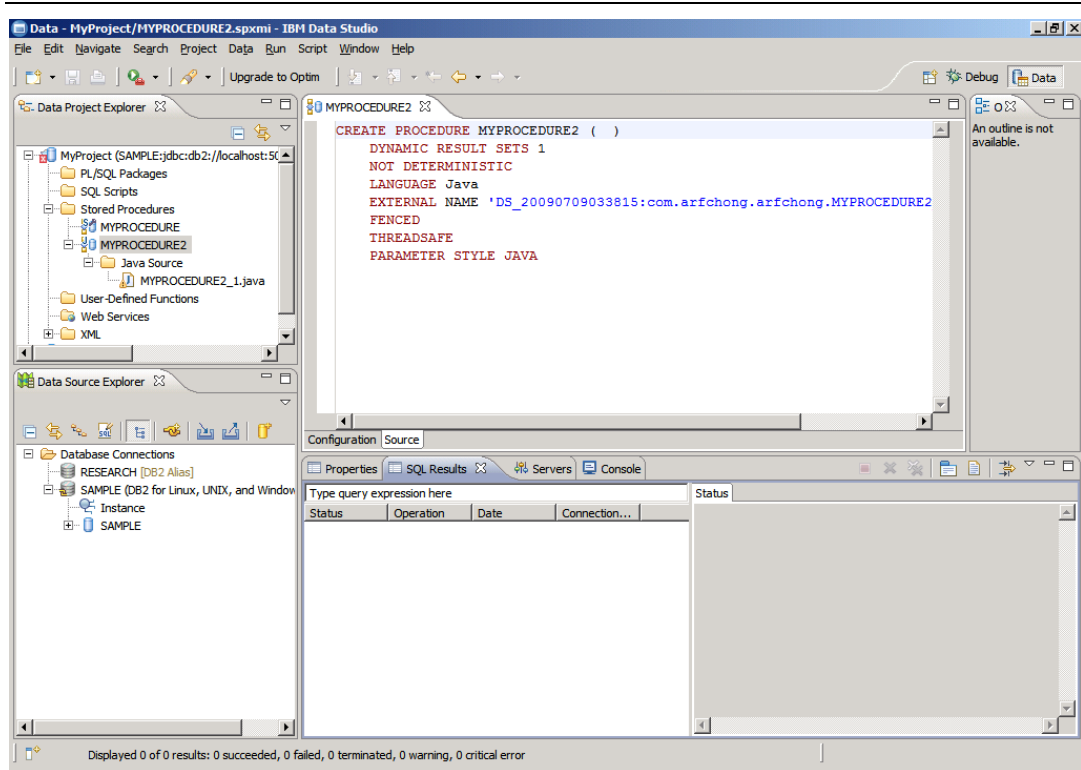


Figure 3.11 - CREATE PROCEDURE definition for a Java stored procedure



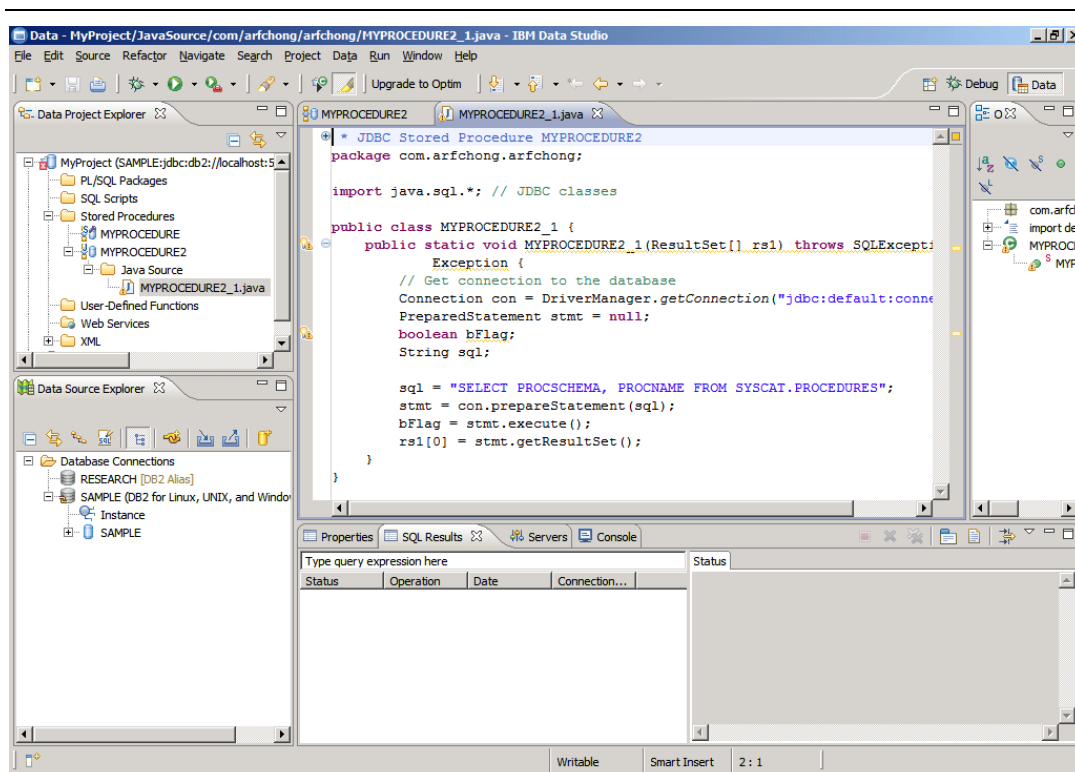


Figure 3.12 - Java source code for a Java stored procedure

### 3.5 User-defined functions: The big picture

A user-defined function (UDF) is a database application object that maps a set of input data values into a set of output values. For example, a function may take a measurement in inches as input, and return the result in centimeters.

DB2 supports creating functions using SQL PL, PL/SQL, C/C++, Java, CLR (Common Language Runtime), and OLE (Object Linking and Embedding). In this book, we focus on SQL PL functions because of their simplicity, popularity, and performance.

**Note:**

Prior to DB2 9.7, UDFs only supported a subset of SQL PL statements known as inline SQL PL. With DB2 9.7, All SQL PL statements are fully supported.

**Note:**

For more information about UDFs watch this video:

<http://www.channeldb2.com/video/video/show?id=807741:Video:4362>

There are four types of functions: scalar, table, row, and column functions. In this book, we focus only on scalar and table functions.

### 3.5.1 Scalar functions

Scalar functions return a single value. Scalar functions cannot include SQL statements that will change the database state; that is, INSERT, UPDATE, and DELETE statements are not allowed. Some built-in scalar functions are SUM(), AVG(), DIGITS(), COALESCE(), and SUBSTR().

DB2 allows you to build customized user-defined functions where you can encapsulate frequently used logic. *Listing 3.9* provides an example of a scalar function.

```
CREATE FUNCTION deptname(p_empid VARCHAR(6))
RETURNS VARCHAR(30)
SPECIFIC deptname
BEGIN ATOMIC
    DECLARE v_department_name VARCHAR(30);
    DECLARE v_err VARCHAR(70);
    SET v_department_name = (
        SELECT d.deptname FROM department d, employee e
        WHERE e.workdept=d.deptno AND e.empno= p_empid);
    SET v_err = 'Error: employee ' || p_empid || ' was not found';
    IF v_department_name IS NULL THEN
        SIGNAL SQLSTATE '80000' SET MESSAGE_TEXT=v_err;
    END IF;
RETURN v_department_name;
END
```

#### Listing 3.9 - An example of a scalar function

In the above listing, the function name is *deptname* and it returns the department number of an employee based on the employee id. A scalar UDF can be invoked using the VALUES statement. It can also be invoked from a SQL statement wherever a scalar value is expected. For example, try the following from the DB2 Command Window or from a Linux or UNIX terminal:

```
db2 "values (deptname ('000300'))"
```

or

```
db2 "select (deptname ('000300')) from sysibm.sysdummy1"
```

Note in the second example that the SYSIBM.SYSDUMMY1 is used. This is a special dummy table with one row and one column. It is used to ensure that only one value is returned. If you try the same SELECT statement with any other table that had more rows, the function would be invoked as many times as the table has.

### 3.5.2 Table functions

Table functions return a table of rows. You can call them using the FROM clause of a query. Table functions, as opposed to scalar functions, can change the database state; therefore, INSERT, UPDATE, and DELETE statements are allowed. Some built-in table functions are SNAPSHOT\_DYN\_SQL() and MQREADALL(). Table functions are similar to views, but since they allow for data modification statements (INSERT, UPDATE, and DELETE) they are more powerful. Typically they are used to return a table and keep an audit record.

*Listing 3.10* provides an example of a table function that enumerates a set of department employees:

```
CREATE FUNCTION getEnumEmployee(p_dept VARCHAR(3))
RETURNS TABLE
  (empno CHAR(6),
   lastname VARCHAR(15),
   firstnme VARCHAR(12))
SPECIFIC getEnumEmployee
RETURN
  SELECT e.empno, e.lastname, e.firstnme
     FROM employee e
     WHERE e.workdept=p_dept
```

#### Listing 3.10 - An example of a scalar function

To test the above function, try the SELECT statement shown in *Figure 3.13* below.

```
SELECT * FROM
TABLE (getEnumEmployee('E01')) T
```

TABLE() function                      alias

**Figure 3.13 – Invoking a table function.**

As shown in the above figure, a table UDF has to be invoked in the FROM clause of an SQL statement since it returns a table. The special TABLE() function must be applied and an alias must be provide after its invocation.

### 3.6 Triggers: The big picture

Triggers are database objects associated with a table that define operations to occur when an INSERT, UPDATE, or DELETE operation is performed on the table. Triggers are

activated automatically. The operations that cause triggers to fire are called triggering SQL statements.

**Note:**

For more information about triggers watch this video:

<http://www.channeldb2.com/video/video/show?id=807741:Video:4367>

new in  
V9.7

**Note:**

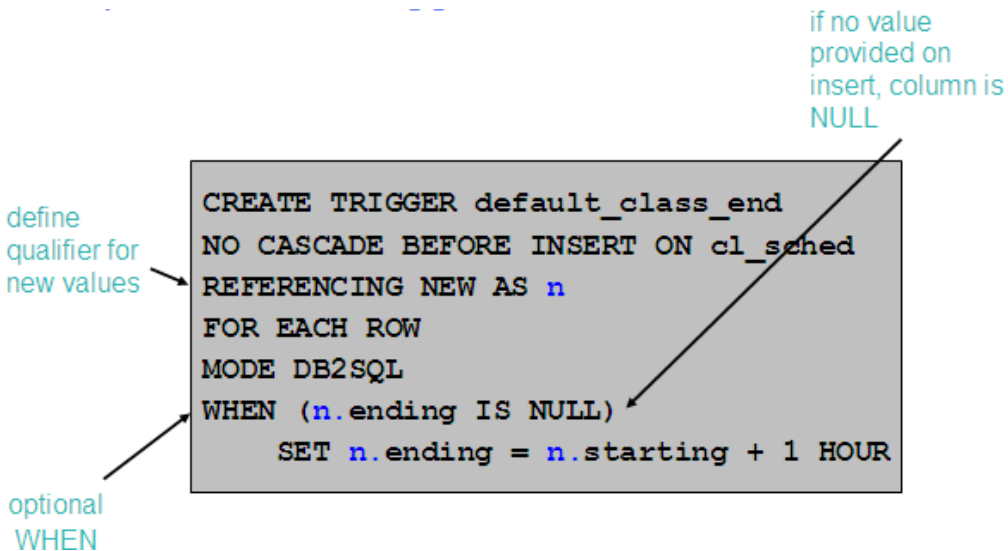
Prior to DB2 9.7, triggers only supported a subset of SQL PL statements known as inline SQL PL. With DB2 9.7, All SQL PL statements are fully supported.

### 3.6.1 Types of triggers

There are three types of triggers: “before” triggers, “after” triggers, and “instead of” triggers.

#### 3.6.1.1 Before triggers

Before triggers are activated before a row is inserted, updated or deleted. The operations performed by this trigger cannot activate other triggers; therefore an INSERT, UPDATE, or DELETE operations are not permitted. An example of a simple before trigger with explanation of some of its syntax is shown in *Figure 3.14*.



**Figure 3.14 – Example of a before trigger**

In the above figure the trigger *default\_class\_end* will be triggered before an INSERT SQL statement is performed on the table CL\_SCHEDULED. This table is part of the SAMPLE database, so you can create and test this trigger yourself while connected to this database.

The variable *n* in the trigger definition represents the new value in an INSERT, that is, the value being inserted. The trigger will check the validity of what is being inserted into the table. If the column ENDING has no value during an INSERT, the trigger will ensure it has the value of the column STARTING plus 1 hour.

*Listing 3.11* shows the statements you can try to test the trigger, and the corresponding output.

```
C:\Program Files\IBM\SQLLIB\BIN>db2 insert into cl_sched (class_code, day,
starting) values ('abc',1,current time)
DB20000I The SQL command completed successfully.
```

```
C:\Program Files\IBM\SQLLIB\BIN>db2 select * from cl_sched
```

CLASS_CODE	DAY	STARTING	ENDING
042:BF	4	12:10:00	14:00:00
553:MJA	1	10:30:00	11:00:00
543:CWM	3	09:10:00	10:30:00
778:RES	2	12:10:00	14:00:00
044:HD	3	17:12:30	18:00:00
abc	1	11:06:53	12:06:53

```
6 record(s) selected.
```

### Listing 3.11 - Testing the before trigger created earlier

In the above listing you can see that there was no value passed for the ENDING column in the INSERT statement; therefore its value is NULL. Also the CURRENT TIME is a special DB2 register returning the time it was invoked. In the example, the current time is 11:06:53, therefore this value is assigned to the STARTING column, while the ENDING column gets what the trigger logic assigns to it, which is 11:06:53 plus 1 hour.

The trigger *validate\_sched* shown below extends the functionality of the *default\_class\_end* trigger previously described to add additional conditions.

```
CREATE TRIGGER validate_sched
NO CASCADE BEFORE INSERT ON cl_sched
REFERENCING NEW AS n
FOR EACH ROW
MODE DB2SQL
BEGIN ATOMIC
-- supply default value for ending time if null
IF (n.ending IS NULL) THEN
    SET n.ending = n.starting + 1 HOUR;
END IF;
```

```
-- ensure that class does not end beyond 9pm
IF (n.ending > '21:00') THEN
    SIGNAL SQLSTATE '80000'
    SET MESSAGE_TEXT='class ending time is beyond 9pm';
ELSEIF (n.DAY=1 or n.DAY=7) THEN
    SIGNAL SQLSTATE '80001'
    SET MESSAGE_TEXT='class cannot be scheduled on a weekend';
END IF;
END
```

### Listing 3.12 - Extending the before trigger created earlier

#### 3.6.1.2 After triggers

After triggers are activated after the triggering SQL statement has executed to successful completion. The operations performed by this trigger may activate other triggers (cascading is permitted up to 16 levels). After triggers support INSERT, UPDATE and DELETE operations. *Listing 3.13* is an example of an after trigger.

```
CREATE TRIGGER audit_emp_sal
AFTER UPDATE OF salary ON employee
REFERENCING OLD AS o NEW AS n
FOR EACH ROW
MODE DB2SQL
    INSERT INTO audit VALUES (
        CURRENT TIMESTAMP, ' Employee ' || o.empno || ' salary changed from '
        || CHAR(o.salary) || ' to ' || CHAR(n.salary) || ' by ' || USER)
```

### Listing 3.13 - An example of an after trigger

In the above listing, the trigger *audit\_emp\_sal* is used to perform auditing on the column SALARY of the EMPLOYEE table. When someone makes a change to this column, the trigger will be activated to write the information about the changed made to the salary into another table called AUDIT. The OLD as o NEW as n line indicates that the prefix o will be used to represent the old or existing value in the table, and the prefix n will be used to represent the new value coming from the UPDATE statement. Thus, o.salary represents the old or existing value of the salary, and n.salary represents the updated value for the column salary data.

#### 3.6.1.3 “Instead of” triggers

*Instead of* triggers are defined on views. Since views are defined dynamically using a SELECT statement that accesses one or more tables, views cannot be updated. However, using this type of trigger, you can give users the illusion that a view can be updated because the logic defined in the trigger is executed instead of the triggering SQL statement. For example, if you perform an update operation on a view, the instead of

trigger will be activated to actually perform the update on the base tables that form the view.

Triggers cannot be created from IBM Data Studio. They can be created from the Control Center or from the Command line tools (DB2 Command Window, Command Line Processor, or the Command Editor).

### 3.7 Data Web services

Data Web services are web services based on database information. Using IBM Data Studio it is very easy to create Data Web services. A web service is like a JEE application (formerly known as J2EE); therefore the Web service needs to be deployed to an application server. In this book we use WebSphere Application Server Community Edition (WAS CE) version 2.1 which is a free application server built on top of Apache Geronimo. WAS CE can be downloaded from this site:

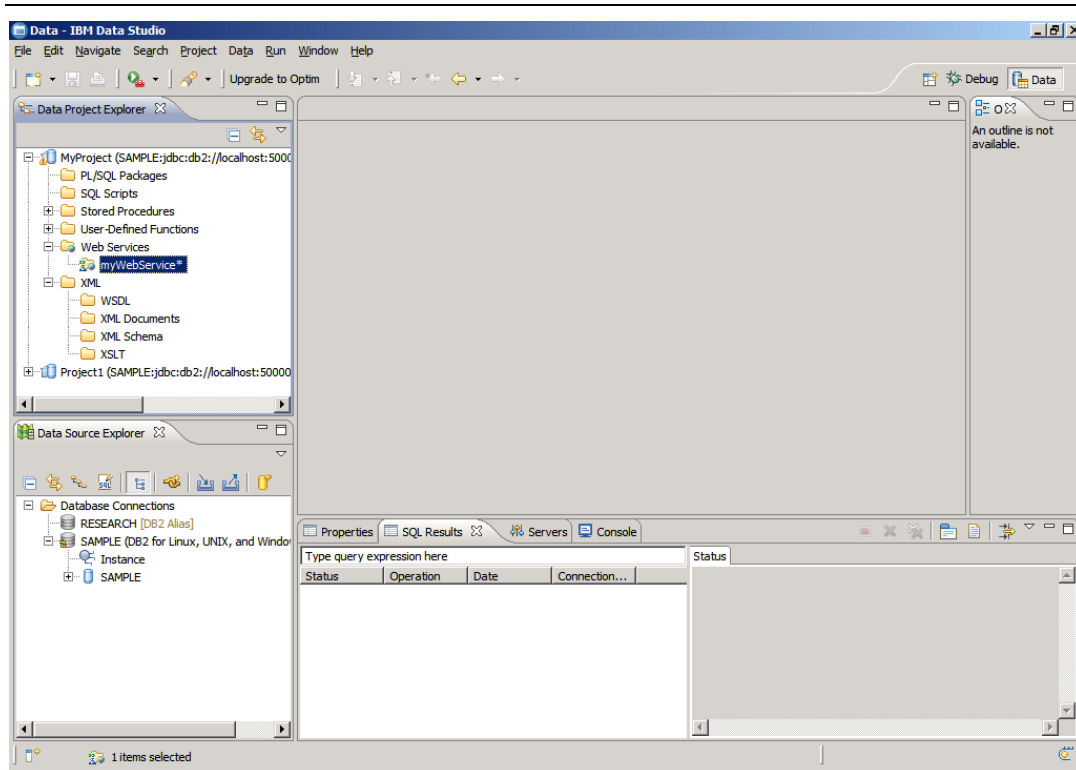
<http://www.ibm.com/developerworks/downloads/ws/wasce/>

WAS CE has a small footprint, and is very easy to install. Ensure WAS CE is installed prior to working with data web services.

**Note:**

For more information about WAS CE, refer to the eBook *Getting started with WAS CE* that is part of this DB2 on Campus free book series.

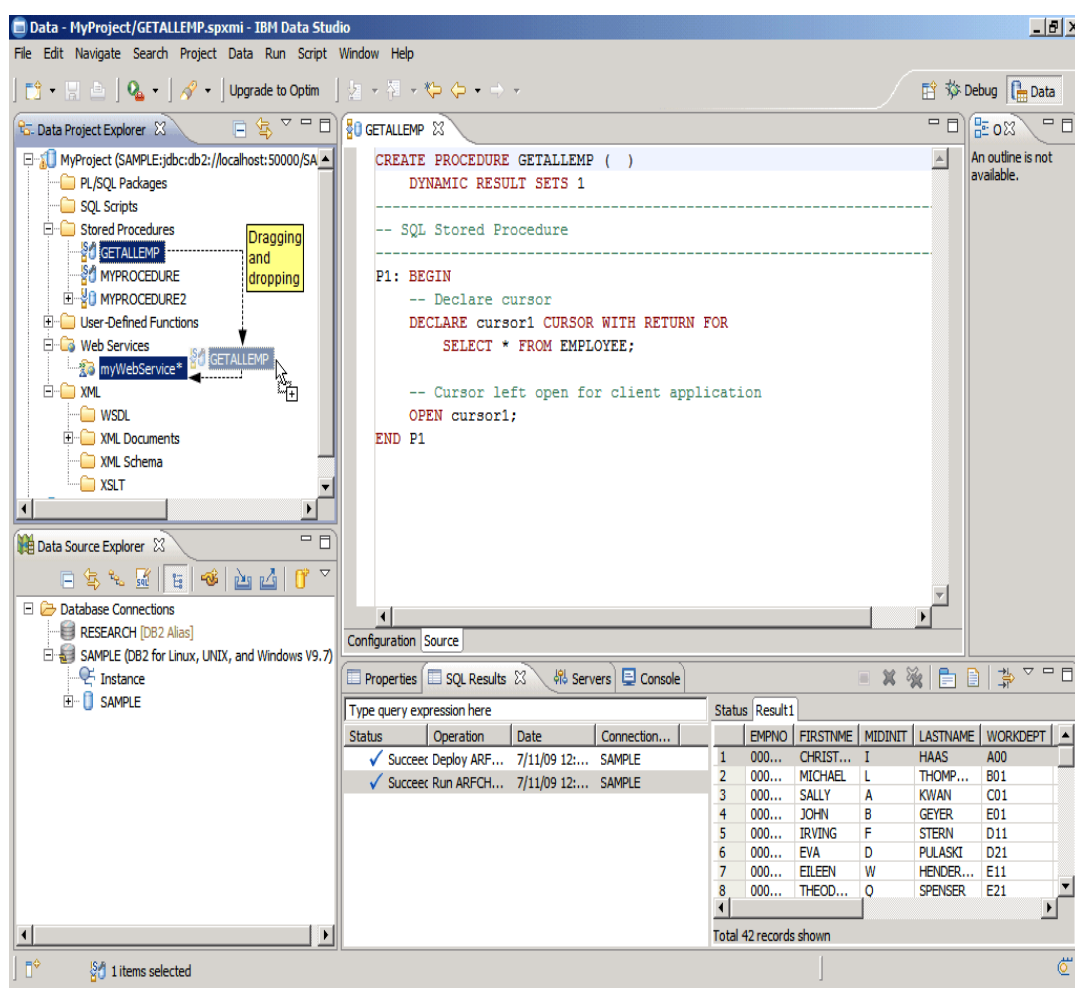
To create a Data Web Service from Data Studio, open or create a new project, and select the *Web Services* folder. Right-click on this folder and choose *New Web Service*. Give a name to the Web service, and click on *Finish*. *Figure 3.15* shows the **MyWebService** created using the steps just explained.



**Figure 3.15 – Creating a data web service**

Though you created a Web service, it currently has no methods or operations. To add operations to the Web service you simply have to drag and drop stored procedures or SQL scripts previously created in Data Studio. For example, take a look at *Figure 3.16* below.

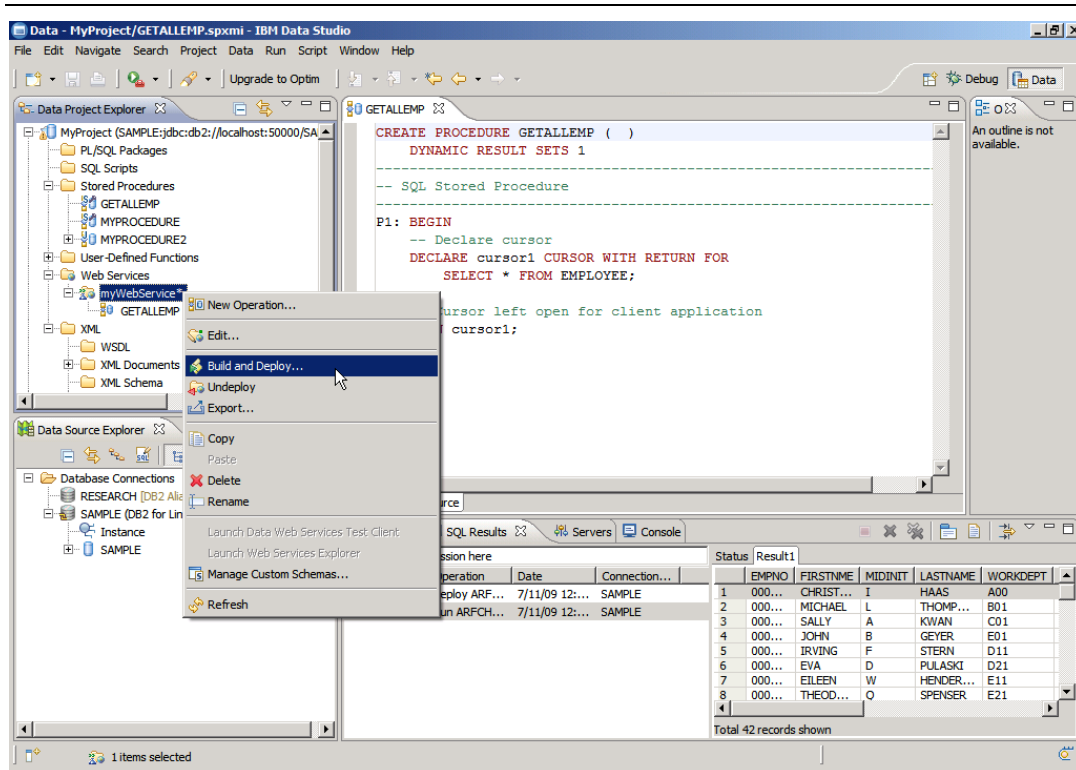




**Figure 3.16 – Dragging and dropping to create a data web service operation**

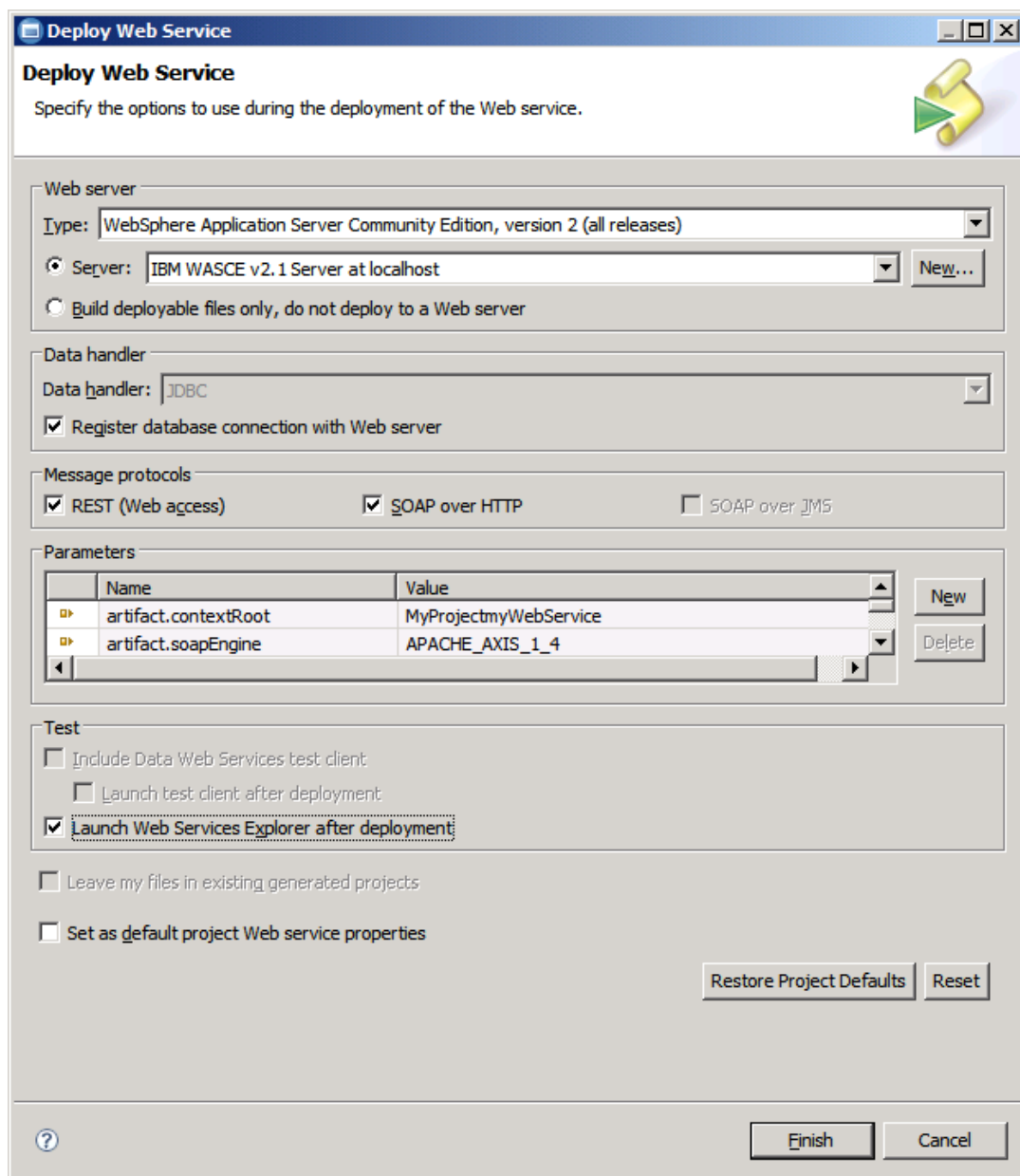
The above figure shows the stored procedure GETALLEMP which returns a cursor based on the statement `SELECT * FROM EMPLOYEE`. After this procedure has been deployed and tested, it is added as an operation to the Web service simply by dragging and dropping it into the web service *myWebService* as highlighted in the figure. A similar procedure can be done with SQL scripts previously created, or even stored procedures already deployed to the database and found from the Data Source Explorer view.

Once the web service has at least one operation, you can build and deploy it by selecting the Web service, right-clicking on it and choosing *Build and Deploy* as shown in Figure 3.17 below.



**Figure 3.17 – Building and deploying a data web service**

After choosing *Build and Deploy*, the *Deploy Web Service* window will appear as illustrated in Figure 3.18.



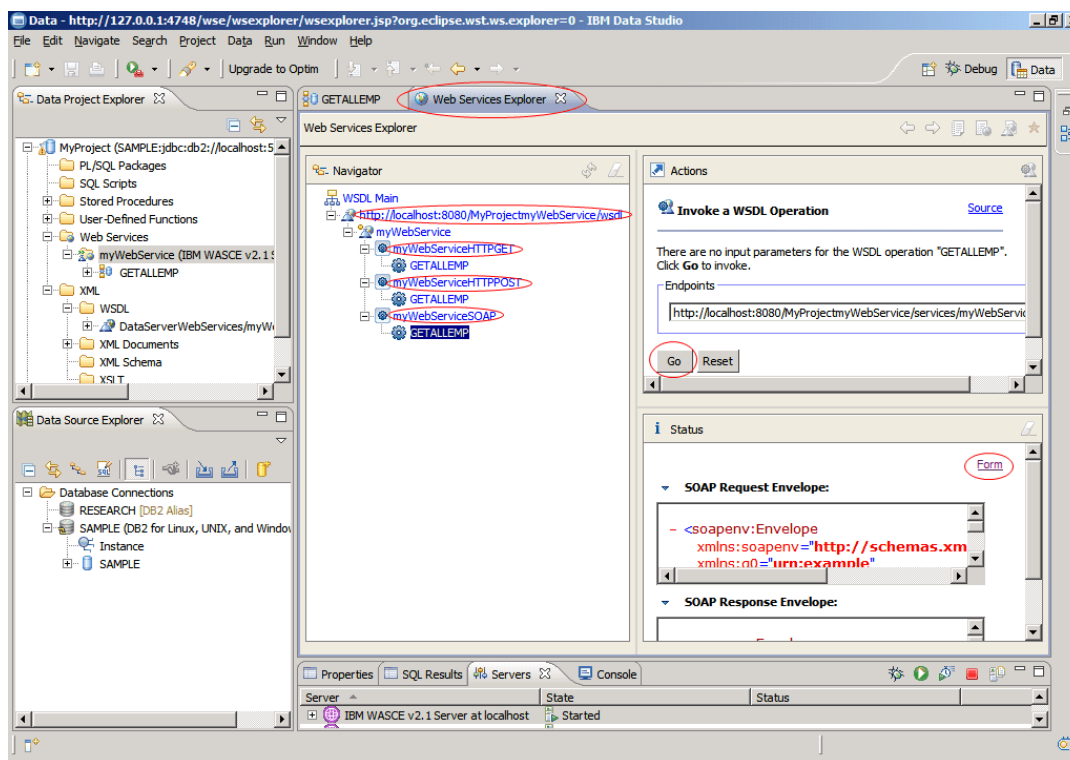
**Figure 3.18 – The Deploy Web Service window**

In the Deploy Web Service window, within the Web Server section, we chose WAS CE 2.1 for the *type* field. Other supported application servers are Apache Tomcat and WebSphere Application Server (WAS). Since we have WAS CE already installed on the same computer as the DB2 data server, we choose the option *Server* rather than *Build deployable files only, do not deploy to a Web server*. This second option creates WAR files you can later transfer to the application server where you want to deploy the Web service. In *Figure 3.20*

above, the application server has already been added to IBM Data Studio. If it had not been added, you need to click the *New* button and take all defaults for most panels. The one thing you do need to specify is where you installed WAS CE. The default installation path on Windows is:

C:\Program Files\IBM\WebSphere\AppServerCommunityEdition

Back in the *Deploy Web Service* window, ensure to click on the *Launch Web Services Explorer after deployment* checkbox. Then click *Finish*. At this point, WAS CE will started, and then the Data Web Service is deployed. This may take approximately 10 to 20 seconds. Once finished the Web Services Explorer will be launched. Figure 3.19 illustrates the Web Services Explorer.



**Figure 3.19 – The Web Services Explorer**

In the above figure, on the left panel of the Web Services Explorer we have a tree which starts with the URL where we can find the Web Services Description Language (WSDL) document. For this particular example, the WSDL is located at:

<http://localhost:8080/MyProjectmyWebService/wsdl>

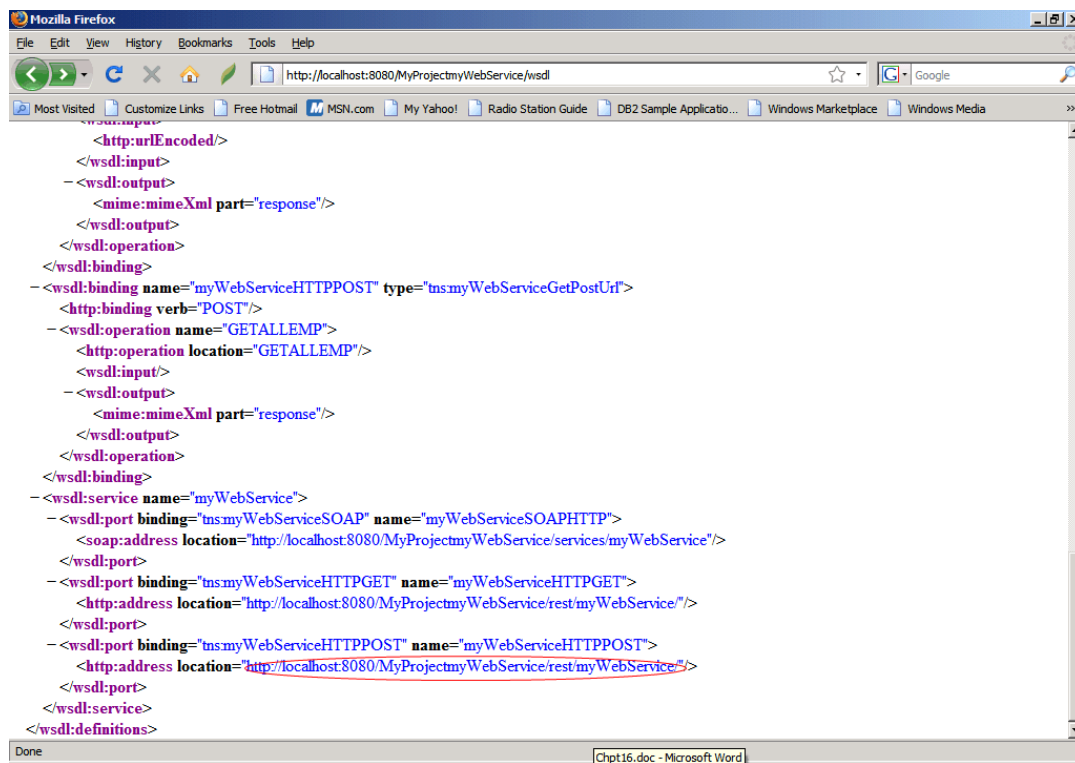
The tree also shows the items:

- myWebServiceHTTPGET
- myWebServiceHTTPGET

- myWebServiceSOAP

The first two items correspond to the REST protocol, and the last one to the SOAP protocol. In *Figure 3.18* under *Message protocols* section, we checked both the REST and SOAP protocols, that's why both type of web services were generated. REST and SOAP are two standards that can be used with Web Services, and it is your choice which want you want to use. IBM Data Studio can generate web services using either or both of them. In *Figure 3.19* the GETALLEMP method under the SOAP version is highlighted, and when you click on *Go* in the right panel you will test this method providing you the output at the bottom of the right panel, and showing you the output as a *Form*. In *Figure 3.19*, we are actually displaying the output as *Source*.

If you would like to invoke the REST version of the GETALLEMP method from a browser, first take a look at the WSDL to see how to invoke the method. If you scroll down the WSDL you will see how to invoke the method. This is illustrated in *Figure 3.20*.

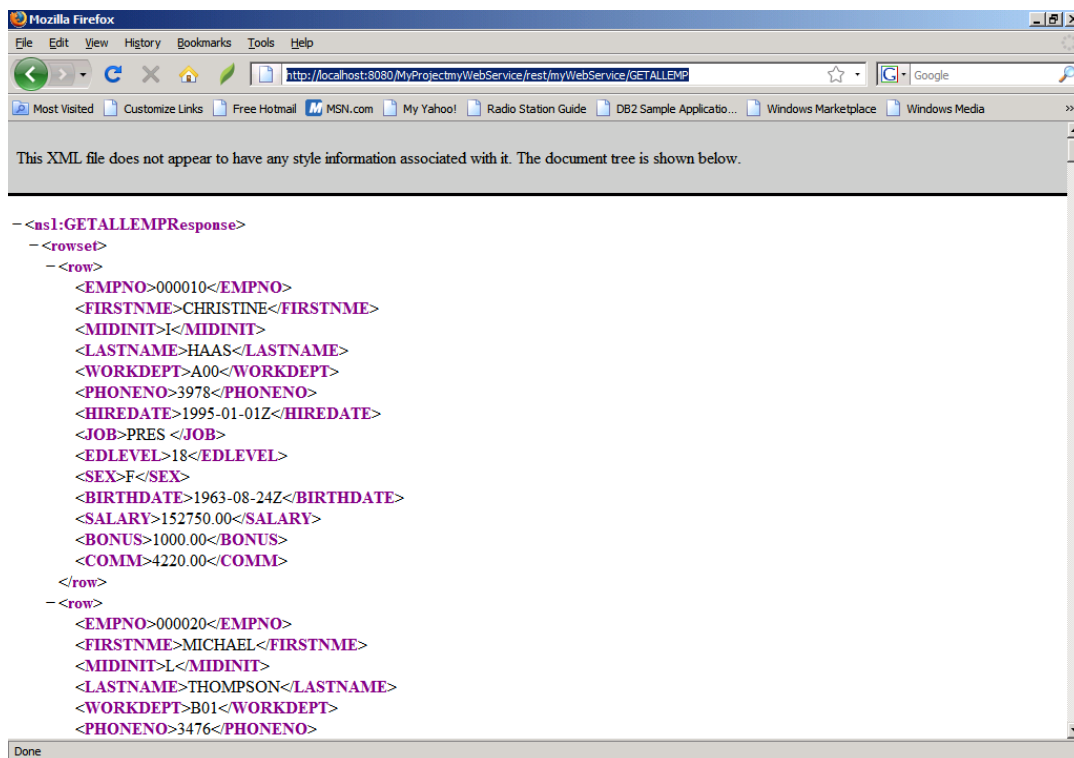


**Figure 3.20 – The WSDL**

In the above figure, we highlighted the URL to use to invoke the Web service. To this URL we need to append the name of the method you wish to execute. The full URL for our example would be:

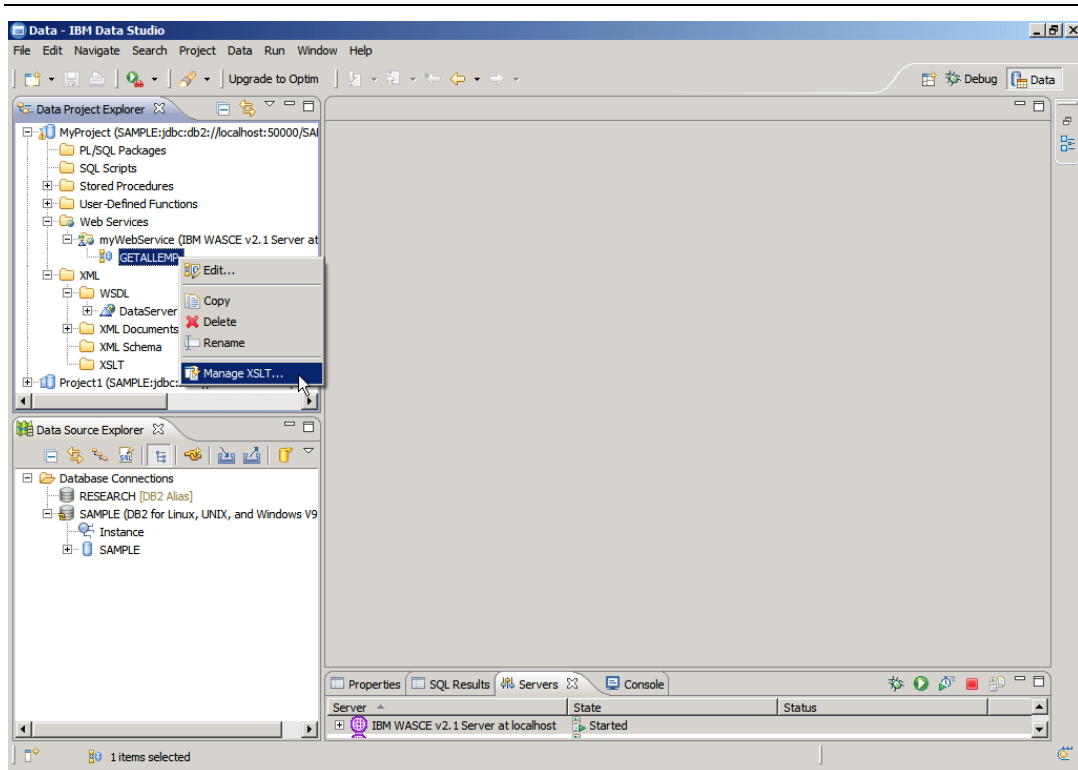
<http://localhost:8080/MyProjectmyWebService/rest/myWebService/GETALLEMP>

Note that the method name is case sensitive. Figure 3.21 illustrates the output of inputting the above URL in a browser.



**Figure 3.21 – Invoking the GETALLEMP method from myWebService**

If you wish, you can also apply an XSLT to this XML output. This can be done from Data Studio by right-clicking on the method in the data Web service and choosing *Manage XSLT* as shown in *Figure 3.22*.



**Figure 3.22 – Applying an XSLT to the output of a Web service method**

In the Configure XSL Transformations window, click on *Browse* and look for an XSL file previously created. For example, Listing 3.14 shows part of the XSL file used in this example.

```

getAllEmp_HTML_Response.xsl - Notepad
File Edit Format View Help
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  xmlns:xalan="http://xml.apache.org/xslt"
  xmlns:tns="http://www.w3.org/1999/xhtml">
/>
  <xsl:output method="html" version="1" encoding="UTF-8" omit-xml-declaration="yes" standalone="yes" indent="yes"
  <xsl:template match="/">
  <html><body>
    <h2>Employees overview</h2>
    <table>
    <tr>
    <td>
      <table border="1">
      <tr>
      <td>EMPNO</td><td>FIRST NAME</td><td>MIDDLE INITIAL</td><td>LAST NAME</td><td>SALARY</td><td>BONUS</td>
      </tr>
      <xsl:for-each select="//row">
      <tr>
      <td>
        <a>
          <xsl:attribute name="href">getEmp?empno=<xsl:value-of select="EMPNO/text()"/></xsl:attribute>
          <xsl:value-of select="EMPNO/text()"/>
        </a>
      </td>
      <td> <xsl:value-of select="FIRSTNAME/text()"/>
      </td>
      <td> <xsl:value-of select="MIDINIT/text()"/>
      </td>
      <td> <xsl:value-of select="LASTNAME/text()"/>
      </td>
      <td>
    </tr>
    </table>
    </td>
  </tr>
  </table>
  </body>
  </html>
  </template>
  </xsl:stylesheet>
  
```

Listing 3.14 - XSL file used in the data Web services example

Figure 3.23 shows how this XSL file is specified.

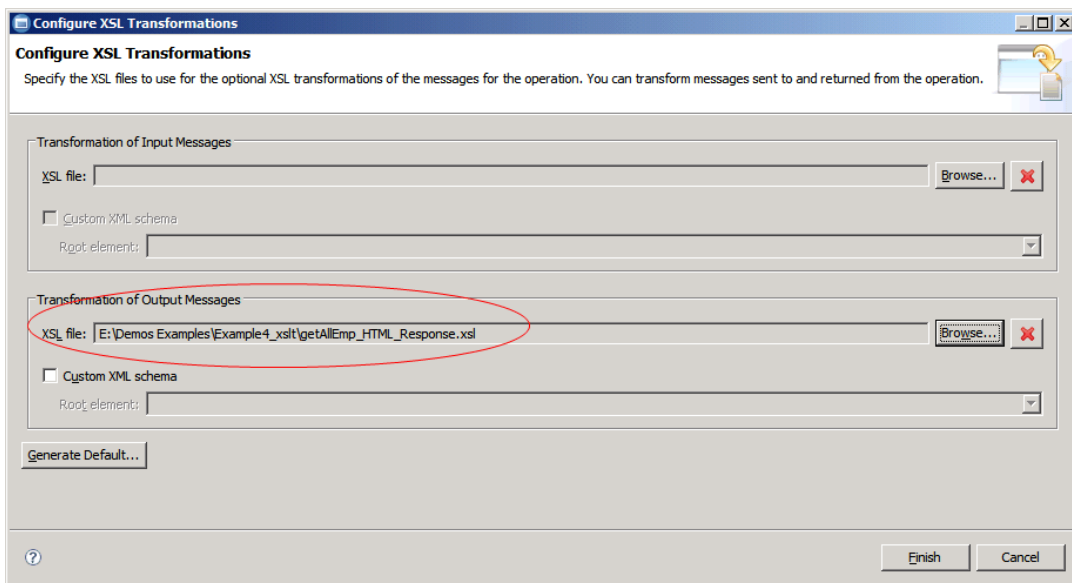


Figure 3.23 – Specifying the XSL file to use

After clicking on Finish, you need to build and deploy the Web service again, and after refreshing the browser with the same URL as in Figure 3.21, you will see an output as illustrated in Figure 3.24.



**Employees Overview**

EMPNO	FIRST NAME	MIDDLE INITIAL	LAST NAME	SALARY	BONUS
000010	CHRISTINE	I	HAAS	152750.00	1000.00
000020	MICHAEL	L	THOMPSON	94250.00	800.00
000030	SALLY	A	KWAN	98250.00	800.00
000050	JOHN	B	GEYER	80175.00	800.00
000060	IRVING	F	STERN	72250.00	500.00
000070	EVA	D	PULASKI	96170.00	700.00
000090	EILEEN	W	HENDERSON	89750.00	600.00
000100	THEODORE	Q	SPENSER	86150.00	500.00
000110	VINCENZO	G	LUCCHESI	66500.00	900.00
000120	SEAN		O'CONNELL	49250.00	600.00
000130	DELORES	M	QUINTANA	73800.00	500.00
000140	HEATHER	A	NICHOLLS	68420.00	600.00
000150	BRUCE		ADAMSON	55280.00	500.00
000160	ELIZABETH	R	PIANKA	62250.00	400.00
000170	MASATOSHI	J	YOSHIMURA	44680.00	500.00
000180	MARILYN	S	SCOUTTEN	51340.00	500.00
000190	JAMES	H	WALKER	50450.00	400.00
000200	DAVID		BROWN	57740.00	600.00
000210	WILLIAM	T	JONES	68270.00	400.00

**Update Bonus:**

Bonus Factor:

Bonus Amount:

**Figure 3.24 – Output after invoking the GETALLEMP method with an XSLT**

**Note:**

For a complete demo of this same Data Web Services example, watch this video:

<http://www.channeldb2.com/video/video/show?id=807741%3AVideo%3A1482>

### 3.8 Exercises

In this exercise, you will create a scalar UDF using IBM Data Studio. This will give you more experience with Data Studio, as well as improving your familiarity with the SQL PL language for user-defined functions.

#### Procedure

1. Open IBM Data Studio (Hint: it is available through the Start menu).
2. Create a new project as described earlier in the chapter that is associated to the EXPRESS database created in the exercises of Chapter 5. Then drill down until you find the *User-Defined Functions* folder.
3. Right-click the *User-Defined Functions* folder. Select *New -> User-defined functions*.

4. For the name of the function use *booktitle*, for the language choose *SQL* which means you will create an SQL PL user-defined function.
5. At this point, you can click the *NEXT* button several times taking all the defaults until you finish creating the UDF. Alternatively, just click on the *Finish* button now.
6. An editor window will be displayed with some sample code. Delete all these code, and replace it with the following.

```
CREATE FUNCTION booktitle(p_bid INTEGER)
RETURNS VARCHAR(300)
-----
SQL UDF (Scalar)
-----
SPECIFIC booktitle
F1: BEGIN ATOMIC
  DECLARE v_book_title VARCHAR(300);
  DECLARE v_err VARCHAR(70);
  SET v_book_title = (SELECT title FROM books WHERE p_bid = book_id);
  SET v_err = 'Error: The book with ID ' || CHAR(p_bid) || '
              was not found.';
  IF v_book_title IS NULL THEN SIGNAL SQLSTATE '80000' SET
    MESSAGE_TEXT=v_err;
  END IF;
RETURN v_book_title;
END
```

7. Build the function by right-clicking on the function name and choosing *Deploy* followed by *Finish* from the *Deploy options* panel.
8. Run the function by right-clicking on the function name and choosing *Run*.
9. Since the function accepts one input parameter, a dialog window appears asking you to fill in a value for the parameter.

Enter the value: 80002

What is the result?

Try again with the value: 1002

What happens this time? (Hint: Look in the *SQL Results* tab).

10. Close IBM Data Studio when you are finished.

### 3.9 Summary

This chapter provided an introduction to data server-side development. We discussed how to create stored procedures, UDFs, and triggers. The discussion was centered on the SQL PL language, and the use of the IBM Data Studio tool. The chapter also discussed how to create Data Web Services based on SQL scripts or stored procedures.

### 3.10 Review questions

1. What are the benefits of stored procedures?
2. Can scalar UDFs be used to write audit information to a table?
3. How can you invoke a scalar UDF?
4. Can a BEFORE trigger be used to UPDATE tables?
5. What is the SPECIFIC keyword used for in a stored procedure?
6. Which of the following is not a valid type of a trigger?
  - A. BEFORE
  - B. PRESENT
  - C. AFTER
  - D. INSTEAD OF
  - E. None of the above
7. Which of the following tools can be used to create a trigger?
  - A. Control Center
  - B. Command Editor
  - C. DB2 Command Window
  - D. IBM Data Studio
  - E. None of the above
8. Why of the following statements correctly invokes the table function getFlights?
  - A. SELECT \* FROM GETFLIGHTS()
  - B. SELECT \* FROM GETFLIGHTS
  - C. SELECT \* FROM TABLE (GETFLIGHTS) A
  - D. SELECT \* FROM TABLE (GETFLIGHTS) AS A
  - E. C and D
9. Which of the following statements is true?
  - A. WSDL stands for Web Services Descriptor List

- B. SOAP and REST Web services both use XML behind the scenes
  - C. Deployed stored procedures cannot be dragged and dropped to a web service in Data Studio to create a new method for that web service
  - D. A and C
  - E. None of the above
10. Which of the following statements is false?
- A. Stored procedures in DB2 can be developed using the C/C++ language
  - B. UDFs only support a subset of the SQL PL language known as inline SQL PL
  - C. A stored procedure can call a UDF
  - D. A trigger can call a stored procedure
  - E. None of the above

# 4

## Chapter 4 – Application development with Java

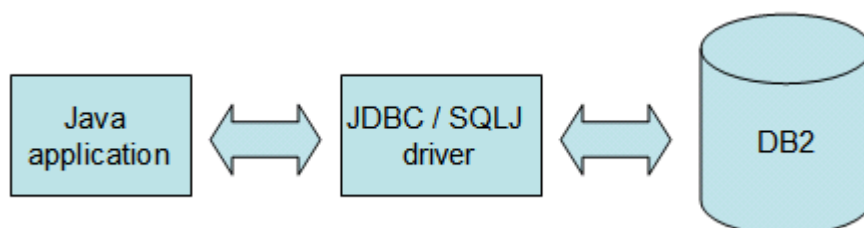
This chapter discusses the basics of application development with Java and DB2. Most Java applications use **Java Database Connectivity (JDBC)** to access databases using dynamic SQL; or SQLJ to access databases using static SQL. These are a set of classes and interfaces written in Java that are vendor-neutral.

In this chapter you will learn about:

- Programming using JDBC
- Programming using SQLJ
- Programming using pureQuery

### 4.1 Java - DB2 applications: The big picture

Developing Java applications that access a database uses the JDBC or SQLJ standard. Support for this standard is provided through a JDBC or SQLJ driver provided by the database vendor as illustrated in *Figure 4.1* where a DB2 data server is used.



**Figure 4.1 - Java applications accessing a DB2 database**

In the figure, a Java application connects to a DB2 data server through the JDBC/SQLJ driver. After a successful connection, SQL or XQuery statements issued from the Java application are passed to the DB2 data server for processing, and then result is returned to the Java application.

JDBC and SQLJ applications can work with minimal modification on data servers that are compliant to the JDBC/SQLJ specifications such as DB2, Informix®, Oracle®, SQL Server®, and so on. DB2 9.7 has support for the JDBC 4.0 specification and earlier.

In the case of DB2, its JDBC and SQLJ driver is included in either of the following:

- Any DB2 data server edition
- IBM Data Server Client
- IBM Data Server Runtime Client
- IBM Data Server Driver for JDBC and SQLJ

Data server editions, clients and drivers were described in *Chapter 1 - What is DB2 Express-C?* Clients and drivers are free of charge. Depending on the type of JDBC driver you use in your application, you may require a client to be installed or not. The different types of JDBC drivers are explained in the next section.

## 4.2 Setting up the environment

Before you can run or develop JDBC or SQLJ applications ensure your environment is correctly set up.

If you would like to develop Java stored procedures and Java user-defined functions which reside on the DB2 server, ensure a JDK is installed in the DB2 server. Fortunately, when you install DB2 Version 9.7, the IBM SDK for Java 6 (also known as JDK 6) is installed by default on all platforms. The location where it is installed is indicated in the database manager configuration (*dbm cfg*) parameter `JDK_PATH`. For example, on Windows the default location is `C:\Program Files\IBM\SQLLIB\java\jdk`. The Java compiler (`javac`) at this location will be used to compile Java stored procedures and Java user-defined functions. The `JVM` that is part of this JDK is used to start the DB2 GUI tools such as the Control Center.

If you would like to develop JDBC/SQLJ applications from the DB2 server; in addition to the JDK, ensure the JDBC/SQLJ driver is setup correctly by adding the correct jar files to the `CLASSPATH`. Similarly, if you would like to develop JDBC/SQLJ application from a client machine, ensure you have installed and setup the JDBC/SQLJ driver correctly. If you are using the JDBC type 2 driver, you also need to install a DB2 client (either the IBM Data Server Client or the IBM Data Server Runtime Client). More details are provided in the next section.

### 4.2.1 DB2 JDBC and SQLJ drivers

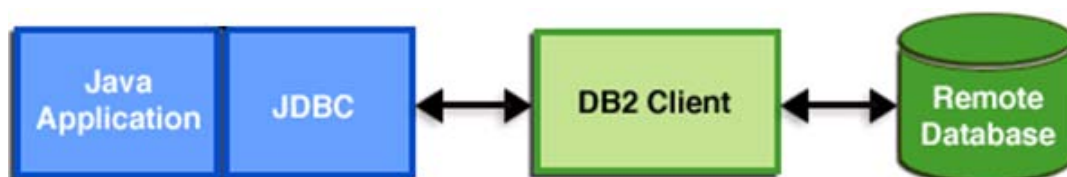
*Table 4.1* describes the different types of JDBC drivers available in the industry today.

Type	Driver name	Description	Provided with DB2?
Type 1	The JDBC-ODBC bridge driver	Through this driver, JDBC access is performed via an ODBC driver.	No

Type 2	The Native-API driver	This driver requires a DB2 client to be installed in the same machine where the JDBC application is running. DB2 provides two type 2 drivers as we will describe later.	Yes
Type 3	The JDBC net pure-java driver	This driver uses a pure Java client and communicates with a net server using a database-independent protocol. The net server then communicates the client's requests to the database. This driver is no longer supported in DB2 in favor of Type 4 drivers.	No
Type 4	The native-protocol pure-java driver.	This is the most flexible JDBC API. This driver converts JDBC calls into the network protocol used by DB2 directly. This allows a direct call from the client machine to the DB2 server without having to install a DB2 client.	Yes

**Table 4.1 - JDBC driver types**

Though there are several types of JDBC drivers, type 2 and type 4 drivers are the most popular and best for performance; therefore, in DB2 9.7, support for other types has been dropped in favor of these two types. Type 2 drivers need to have a DB2 client installed on the machine where the Java application is running. The type 2 driver uses the DB2 client to establish communication to the database as depicted in *Figure 4.2*.



**Figure 4.2 - A Java application using the JDBC type 2 driver**

Type 4 is a pure java client, so there is no need for a DB2 client, but the driver must be installed on the machine where the JDBC application is running. *Figure 4.3* illustrates a JDBC application using the type 4 driver.



**Figure 4.3 – A JDBC application using the type 4 driver**

Table 4.2 provides more details about the DB2 JDBC and SQLJ drivers.

Driver Type	Driver Name	Packaged as	JDBC specification supported	Minimum level of SDK for Java required
Type 2	DB2 JDBC Type 2 Driver for Linux, UNIX and Windows ( <b>Deprecated*</b> )	db2java.zip	JDBC 1.2 and JDBC 2.0	1.4.2
Type 2 and Type 4	IBM Data Server Driver for JDBC and SQLJ	db2jcc.jar and sqlj.zip	JDBC compliant	3.0 1.4.2
		db2jcc4.jar and sqlj4.zip	JDBC 4.0 and earlier	6

**Table 4.2 - DB2 JDBC and SQLJ drivers**

\* Deprecated means it is still supported, but no longer enhanced

As you can see from Table 4.2, Type 2 is provided with two different drivers; however the DB2 JDBC Type 2 Driver for Linux, UNIX and Windows, packaged as `db2java.zip`, is deprecated.

The IBM Data Server Driver for JDBC and SQLJ packaged as `db2jcc.jar` (`com.ibm.db2.jcc`) includes support for both, the type 2 and type 4 drivers. The choice of driver is determined based on the syntax used to connect to the database in your Java program: If a hostname or IP address, and a port are included in the connection string, then type 4 is used, otherwise, type 2 is used. This is discussed in more detail in a later section of this chapter. The IBM Data Server Driver for JDBC and SQLJ has been optimized to access all DB2 servers in all platforms including the mainframe.

When you install a DB2 server, a DB2 client or the IBM Data Server Driver for JDBC and SQLJ, the `db2jcc.jar` and `sqlj.zip` files compliant with JDBC 3.0 are automatically added to your **CLASSPATH**. If you would like to use the JDBC 4.0 specification, make sure to replace `db2jcc.jar` and `sqlj.zip` with `db2jcc4.jar` and `sqlj4.zip` respectively in the **CLASSPATH**.

**Note:**



If you are new to the Java programming language and the concepts of JVM, JRE, or JDK; review the free e-book *Getting Started with Java* that is part of this book series.

### 4.3 JDBC Programming

Developing a JDBC program consists of the following steps:

1. Connect to the database using either JDBC type 2 or type 4
2. Execute SQL statements
3. Receive results
4. Handle SQL errors and warnings
5. Close the connection

We discuss each of these steps in more detail in the next sections. *Figure 4.4* provides a summary about the steps, interfaces, classes and methods that are used in a JDBC program. These are also discussed in more detail in the next sections.

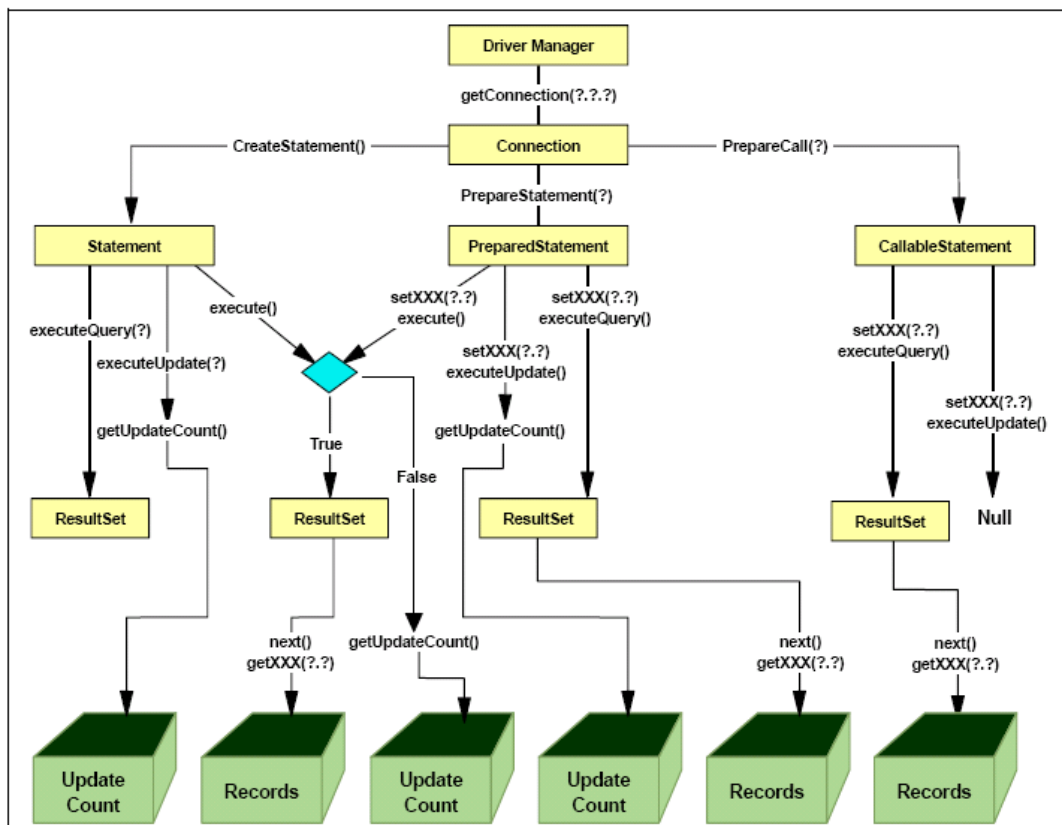


Figure 4.4 - The JDBC Programming steps, objects and methods

**Note:**

Several examples and figures used in this section are taken from the IBM redbook *DB2 Express-C: The Developer Handbook for XML, PHP, C/C++, Java, and .NET* (SG24-7301-00). See the Resources section in this book for more information.

**4.3.1 Connecting to a DB2 database**

This section shows you how to connect to a database using JDBC Type 2, and JDBC Type 4. Let's examine the code snippet shown in *Listing 4.1* below for a connection using JDBC Type 2.

```
(1) import java.sql.*;

class myprg {
    public static void main (String argv[]){
        try {
            Connection con = null;
(2)      Class.forName("com.ibm.db2.jcc.DB2Driver");
(3)      String url = "jdbc:db2:SAMPLE";
            if (argv.length == 2){
                String userID = argv[0];
                String passwd = argv[1];
(4)      con = DriverManager.getConnection(url,userID,passwd);
            }
            else
                {throw new Exception
                    ("\n Usage: java myprg userID password\n");
                }
        }
    }
}
...

```

**Listing 4.1 - Connecting to a DB2 Database using JDBC Type 2**

Let's review each of the items shown in *Listing 4.1*:

- (1) This statement imports the `java.sql` package, which contains the JDBC core API.
- (2) This statement loads the driver classes from the IBM Data Server Driver for JDBC and SQLJ (`db2jcc.jar/db2jcc4.jar/sqlj.zip/sqlj4.zip`). The `forName` method takes a string argument whose value is the name of the class which implements the interfaces defined in `java.sql` package. In this case the class name is `"com.ibm.db2.jcc.DB2Driver"`. If you expand the `db2jcc.jar` file, you'll see on Windows:

```
C:\Program
Files\IBM\SQLLIB\java\db2jcc\com\ibm\db2\jcc\DB2Driver.class
```

In the case of the deprecated DB2 JDBC Type 2 Driver for Linux, UNIX and Windows, the class name to use in the `forName` method would be `"COM.ibm.db2.jdbc.app.DB2Driver"`. If you unzip `db2java.zip` on Windows you'll see:

```
C:\Program
Files\IBM\SQLLIB\java\db2java\COM\ibm\db2\jdbc\app\DB2Driver.
class
```

- (3) In this line, we initialize the URL and choose to connect to the `SAMPLE` database. In the syntax of the URL we are not including the host name or the port number; therefore, this means Type 2 is being used. Type 2 needs a DB2 client to be installed, and use it to configure the connectivity to the `SAMPLE` database.
- (4) In the case two arguments are passed to the program (`userID` and `password`), these will be used to get the connection; otherwise the program throws an exception. `DriverManager.getConnection(url,userID,passwd)` can also be called without a `userID` and `passwd` as follows: `DriverManager.getConnection(url)`. In this case the user ID logged on to the system would be used. For Type 4, as we will see next, this will not work, as this connection is taken as a remote TCPIP connection and DB2 needs a user ID and password for all remote connections.

Now let's take a look at the same code snippet as in *Listing 4.1*, but using a JDBC Type 4 connection. This is illustrated in *Listing 4.2*

```
import java.sql.*;

class myprg {
public static void main (String argv[]){
    try {
        Connection con = null;
        Class.forName("com.ibm.db2.jcc.DB2Driver");
(1)      String url = "jdbc:db2://168.100.10.1:50000/SAMPLE";
        if (argv.length == 2){
            String userID = argv[0];
            String passwd = argv[1];
            con = DriverManager.getConnection(url,userID,passwd);
        }
        else
        { throw new Exception
          ("\n Usage: java myprg userID password\n");
        }
    }
}
```

...

**Listing 4.2 - Connecting to a DB2 Database using JDBC Type 4**

*Listing 4.2* shows the exact same code snippet as in *Listing 4.1*, but the URL has been changed to use the Type 4 syntax:

```
"jdbc:db2://<IP address or hostname>:<DB2 Instance port number>/<dbname>"
```

- (1) In *Listing 4.2*, the fictitious IP address 168.100.10.1 was used. The DB2 instance port number is 50000, and the database name to connect to is **SAMPLE**. To test a connection when you are not connected to a network you can always use `localhost` or the loopback IP address 127.0.0.1 to point to yourself.

**Note:**

Most of the code snippets shown in this chapter are extracted from the program `myprg.java` which is included in the `Exercise_Files_DB2_Application_Development.zip` file with this book. You can test each code snippet by commenting out the appropriate section in the program.

After a `Connection` object is created, a `Statement`, `PreparedStatement`, or `CallableStatement` object can be created with the methods described in *Table 4.3*

Method	Object created	Description
<code>createStatement</code>	<code>Statement</code> object	A <code>Statement</code> object can be used to execute SQL that does not use parameter markers.
<code>prepareStatement</code>	<code>PreparedStatement</code> object	A <code>PreparedStatement</code> object can be used to execute SQL that uses parameter markers.
<code>prepareCall</code>	<code>CallableStatement</code> object	A <code>CallableStatement</code> object can be used to call a stored procedure.

**Table 4.3 - Methods of the Connection object to create different types of Statement object**

**4.3.2 Executing SQL statements**

This section describes how to declare host variables, and execute SQL statements using the `Statement`, `PreparedStatement` and `CallableStatement` interfaces.

**4.3.2.1 Declaring host variables**

Host variables follow normal Java variable syntax. Some data types that map to database data types in Java applications can be seen in *Figure 4.5* extracted from the DB2 Information Center.

Java data type	Database data type
short	SMALLINT
boolean <sup>1</sup> , byte <sup>1</sup> , java.lang.Boolean	SMALLINT
int, java.lang.Integer	INTEGER
long, java.lang.Long	BIGINT <sup>10</sup>
float, java.lang.Float	REAL
double, java.lang.Double	DOUBLE
java.math.BigDecimal	DECIMAL(p,s) <sup>2</sup>
java.math.BigDecimal	DECFLOAT(n) <sup>3,4</sup>
java.lang.String	CHAR(n) <sup>5</sup>
java.lang.String	GRAPHIC(m) <sup>6</sup>
java.lang.String	VARCHAR(n) <sup>7</sup>
java.lang.String	VARCHAR(n) <sup>8</sup>

**Figure 4.5 - Mapping of Java data types with DB2 data types**

The full mapping list can be found at:

<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.java.doc/doc/rjvjdata.html>

The **DCLGEN** (Declarations generator) utility in DB2 allows you to create structures for host variables. The languages supported are C, Java, COBOL, and FORTRAN.

For example, to generate the declaration statements for the table *employee* in the **SAMPLE** database for the Java language you can use:

```
db2dclgn -D sample -T employee -L Java
```

The output would be stored in a file *employee.java* with content as shown in *Listing 4.3* below.

```
...
java.sql.Date      hiredate;
java.lang.String   job;
short              edlevel;
java.lang.String   sex;
java.sql.Date      birthdate;
java.math.BigDecimal salary;
```

**Listing 4.3 - Output of DCLGEN for the employee table using the Java language**

#### 4.3.2.2 The Statement interface

A class implementing the `Statement` interface is used to execute an SQL statement which does not contain parameter markers. A `Statement` object is created with the `createStatement` method from a `Connection` object.

Table 4.4 shows the different methods applicable to the `Statement` object to execute a query.

Method	Description
<code>executeQuery</code>	Use it when a result set is expected, for example, a <code>SELECT</code> statement. It returns a <code>ResultSet</code> object.
<code>executeUpdate</code>	Use it for updates of database contents, for example, <code>INSERT</code> , <code>UPDATE</code> , and <code>DELETE</code> . It returns an integer with the number of rows affected.
<code>execute</code>	Use it when you don't know until runtime whether the statement executed is a <code>SELECT</code> or an <code>UPDATE</code> statement. It returns <code>true</code> if the result of the SQL is a result set, <code>false</code> if it's an update count. Use this method in conjunction with <code>getResultSet</code> or <code>getUpdateCount</code> methods.

**Table 4.4 - Methods of the Statement object**

Listing 4.4 provides a code snippet that illustrates the use of a `Statement` object, and the `executeQuery` method.

```

...
(1) Statement stmt = con.createStatement();
(2) ResultSet rs = stmt.executeQuery
    ("SELECT EMPNO, FIRSTNME, LASTNAME " +
     " FROM EMPLOYEE " +
     " WHERE SALARY > 80000" );
(3) while ( rs.next() ) {
    System.out.println("Empno = " + rs.getString(1) +
                      " Full name = " + rs.getString(2) +
                      " " + rs.getString(3));
}
(4) rs.close();
(5) stmt.close();
(6) con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
} }

```

**Listing 4.4 - Statement object: Performing a SELECT with executeQuery**

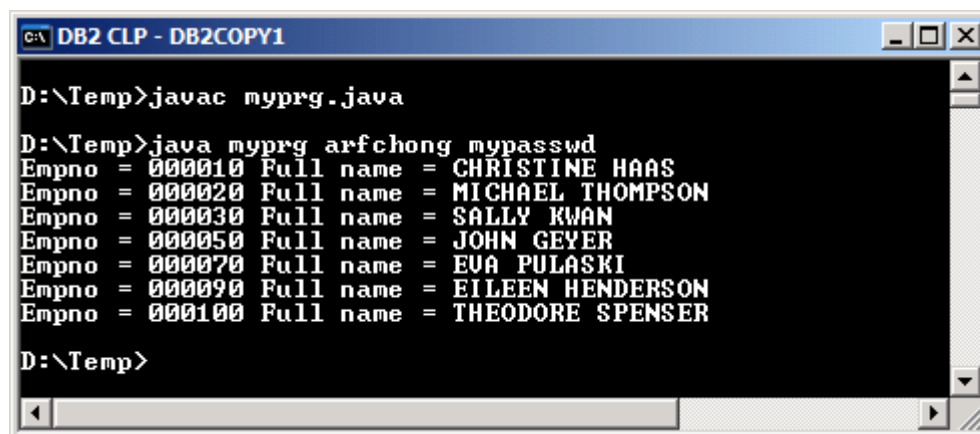
The three dots at the beginning indicates this is part of a program, not all the listing is being shown. Next each line marked with a number is explained as follows:

- (1) An object of `Statement` (or class implementing the `Statement` interface) can be used to execute the SQL statement which does not contain parameter markers. A `Statement` object can be created from the `Connection` object using `createStatement` method.
- (2) The `ResultSet` object maintains a cursor to the current row of the result set of a query. The `executeQuery` method allows you to execute a query (`SELECT`). If you want to update, delete, or insert, use the `executeUpdate` method as we will see later.
- (3) The cursor can be advanced to the next row by using the `next` method of the `ResultSet` object. The cursor by default can only be moved forward and is read-only.
- (4) Closing the result set.
- (5) Closing the statement.
- (6) Closing the connection.

If you would like to test the above code snippet, edit the `myprg.java` program (accompanying this book) appropriately. The program includes the connection statements shown in *Listing 4.2*. The program would be compiled and executed as shown below.

```
javac myprg.java
java myprg <userid> <password>
```

The output would look as shown in *Figure 4.6* where the `userID` is `arfchong`, and the password is `mypasswd`:



```
C:\ DB2 CLP - DB2COPY1
D:\Temp>javac myprg.java
D:\Temp>java myprg arfchong mypasswd
Empno = 000010 Full name = CHRISTINE HAAS
Empno = 000020 Full name = MICHAEL THOMPSON
Empno = 000030 Full name = SALLY KWAN
Empno = 000050 Full name = JOHN GEYER
Empno = 000070 Full name = EVA PULASKI
Empno = 000090 Full name = EILEEN HENDERSON
Empno = 000100 Full name = THEODORE SPENSER
D:\Temp>
```

Figure 4.6 - Executing the `myprg.java` program

The next code snippet shown in *Listing 4.5* below provides an example of using a `Statement` object, and the `executeUpdate` method.

```
Statement updStmt = con.createStatement();
(1) int numRows = updStmt.executeUpdate
    ("UPDATE EMPLOYEE " +
     " SET FIRSTNME = 'Raul', " +
     "     LASTNAME = 'Chong' " +
     " WHERE EMPNO = '000010' ");
System.out.println("Number of rows updated " + numRows);
```

**Listing 4.5 - Statement object: Performing an UPDATE with executeUpdate**

In the above listing, in line (1):

The `executeUpdate` method is used to perform a SQL UPDATE operation, which returns an integer with the number of rows affected. The above code snippet is part of the `myprg.java` program. You can run it as follows after commenting out the appropriate sections in the program:

```
javac myprg.java
java myprg <userid> <password>
```

And this would be the output:

```
Number of rows updated: 1
```

The examples in *Listing 4.6* and *Listing 4.7* are similar to the previous *Listing 4.5*, but in this case, the query is saved on a string called `query` first, and we are using the `executeUpdate` method to perform an `INSERT` and a `DELETE` respectively.

```
String query = null;
query = "INSERT INTO employee (EMPNO, " +
        "FIRSTNME, LASTNAME, EDLEVEL, SALARY)" +
        "VALUES ('099999', 'Jin', 'Xie', 25, 90000)";
Statement stmt = con.createStatement();
int numRows = stmt.executeUpdate( query );
System.out.println("Number of rows inserted: " + numRows);
stmt.close();
```

**Listing 4.6 - Statement object: Performing an INSERT with executeUpdate**

```
String query = null;
query = "DELETE FROM employee where empno = '000999'";
Statement stmt = con.createStatement();
int numRows = stmt.executeUpdate( query );
```



```
System.out.println("Number of rows deleted: " + numRows);
stmt.close();
```

#### Listing 4.7 - Statement object: Performing a DELETE with executeUpdate

The next code snippet shown in *Listing 4.8* is an example of using the `execute` method. As stated earlier, this method is used when you don't know until runtime if you are performing a `SELECT` or an update, where an update refers to an SQL `UPDATE`, `INSERT`, or `DELETE`.

```
String passedStmt = "SELECT firstname, lastname " +
                    "FROM employee " +
                    "WHERE salary > 80000";
Statement stmt    = con.createStatement();
ResultSet rs      = null;
int numRows      = 0;
(1) if (stmt.execute(passedStmt)){
    rs = stmt.getResultSet();
    while ( rs.next() ) {
        System.out.println("Full name = " + rs.getString(1) +
                           " " + rs.getString(2));
    }
    rs.close();
}
else {
    numRows = stmt.getUpdateCount();
    System.out.println("Number of rows updated: " + numRows);
}
```

#### Listing 4.8 - Statement object: Performing a SELECT or UPDATE with execute

In the above listing, in line (1):

If the statement passed (*passedStmt* in this example) is a `SELECT`, the `execute` method will return `true`. If it was an `UPDATE/INSERT/DELETE`, it'd return `false`. In this particular example we hard-coded the statement passed; however, it could have been implemented as an argument.

To test this using the `myprg.java` program, comment out the corresponding section in the program, and run these commands:

```
javac myprg.java
java myprg <userid> <password>
```

The output would look like:

```
Full name = Raul Chong
Full name = MICHAEL THOMPSON
```

```
Full name = SALLY KWAN
Full name = JOHN GEYER
Full name = EVA PULASKI
Full name = EILEEN HENDERSON
Full name = THEODORE SPENSER
```

If you modify the program so the passed statement is an **UPDATE** operation, you can test compiling and running the program again, and would get this result:

```
Number of rows updated: 1
```

#### 4.3.2.3 The PreparedStatement interface

A class implementing the **PreparedStatement** interface can be used to run queries which can contain **parameter markers**. A parameter marker is a question mark (?) that appears in a dynamic statement string and can appear where a variable could appear. **PreparedStatement** extends the **Statement** interface.

As indicated earlier, the **prepareStatement** method of the **Connection** object is used to create a **PreparedStatement** object.

If the SQL statement contains parameter markers, the values for these parameter markers need to be set using setter methods before executing the statement. Setter methods of a **PreparedStatement** object look like "setXXX", where XXX denotes the data type of the parameter marker. For example, **setInt**, **setString**, **setDouble**, **setBytes**, **setClob**, **setBlob**

After setting the parameter values, use the **executeQuery**, **executeUpdate**, or **execute** methods based on the SQL type.

*Listing 4.9* provide a sample code snippet using **PreparedStatement** and a **SELECT**.

```
(1) PreparedStatement pstmt = con.prepareStatement
    ("SELECT firstme, " +
     " lastname " +
     "FROM employee WHERE salary > ? ");
(2) pstmt.setInt(1,80000);
(3) ResultSet rs = pstmt.executeQuery();
    while ( rs.next() ) {
        System.out.println("Full name = " + rs.getString(1) +
                            " " + rs.getString(2));
    }
(4) rs.close();
(5) pstmt.close();
```

#### Listing 4.9 - PreparedStatement object: Performing a SELECT with executeQuery

In the above listing:

- (1) A prepared statement is created from a **SELECT**, where the parameter marker (?) is used for the salary.
- (2) Since the **salary** column is defined as **INTEGER**, we use the setter method **setInt**. In this particular example we hardcode the value to 80000. The "1" in `pStmt.setInt(1,80000)` represents the first parameter marker.
- (3) After the setter methods have been used to set values, we use **executeQuery** in this case since it's a **SELECT** statement, and assign it to a result set.
- (4) Close the result set
- (5) Close the **PreparedStatement** object *pStmt*.

This next sample code snippet shown in *Listing 4.10* provides a similar example as the previous listing, but this time it is using a **PreparedStatement** with an update, and using **executeUpdate**.

```
(1) PreparedStatement pStmt = con.prepareStatement
    ("UPDATE employee " +
     " SET salary = ? " +
     " WHERE empno = ? ");
(2) pStmt.setInt    (1,85000);
(3) pStmt.setString(2,"000010");
(4) int numRows = pStmt.executeUpdate();
    System.out.println("Number of rows updated: " + numRows);
    pStmt.close();
```

#### Listing 4.10 - PreparedStatement object: Performing an UPDATE with executeUpdate

In the above listing:

- (1) A prepared statement is created from an **UPDATE**, where one parameter marker is used for the salary, and another one for the **empno** column.
- (2) Since the **salary** column is defined as **INTEGER**, we use the setter method **setInt**. In this particular example we hardcode the value to 85000. The "1" in `pStmt.setInt(1,85000)` represents the first parameter marker.
- (3) The **empno** column is defined as a string (**CHAR** in the database); therefore, we use the setter method **setString**.
- (4) After the setter methods have been used to set values, we use **executeUpdate** in this case since it's an **UPDATE** statement, and obtain the number of rows affected.

The next sample code snippet shown in *Listing 4.11* provides a similar example as the previous listing, but this time it is using a **PreparedStatement** with a **SELECT**, and using **execute**.

```
(1) String passedStmt = "SELECT firstnme, lastname " +
    "FROM employee " +
    "WHERE salary > ?";
```

```
PreparedStatement pstmt = con.prepareStatement(passedStmt);
pstmt.setInt (1,85000);
ResultSet rs = null;
int numRows = 0;
(2) if (pstmt.execute())
    {
        rs = pstmt.getResultSet();
        while ( rs.next() ) {
            System.out.println("Full name = " + rs.getString(1) +
                               " " + rs.getString(2)); }

        rs.close();
    }
else
    {
        numRows = pstmt.getUpdateCount();
        System.out.println("Number of rows updated: " + numRows); }
pstmt.close();
```

#### Listing 4.11 - PreparedStatement object: Performing a SELECT with execute

In the listing above:

- (1) A prepared statement is created from a **SELECT**, where a parameter marker is used for the salary.
- (2) Use the **execute** method when you don't know whether the statement to be executed is a query (**SELECT**) or an update (**UPDATE**, **INSERT**, **DELETE**). If the statement passed (*passedStmt* in this example) is a **SELECT**, the execute method will return **true**. If it was an **UPDATE/INSERT/DELETE**, it'd return **false**. In this particular example we hard-coded the passed statement. It could have been implemented as an argument.

#### 4.3.2.4 The CallableStatement interface

A class implementing the **CallableStatement** interface can be used to call a stored procedure. **CallableStatement** extends the **PreparedStatement** interface. Use the **prepareCall** method of the **Connection** object to create a **CallableStatement** object.

A **CallableStatement** can have three types of parameters: **IN**, **OUT**, **INOUT**. The value for **IN** and **INOUT** parameters must be set using setter methods (**setXXX**) before executing the **CallableStatement**. In the same way, **OUT** and **INOUT** parameters should be registered (using **registerOutParameter** methods) to the database before executing the statement.

The **CallableStatement** can be executed using **executeUpdate**, **executeQuery**, and **execute** methods. The usage of these three methods is described below:

- **executeUpdate**: When no result set is expected as the output of the call.

- **executeQuery**: When a single result set is expected as the output of the call.
- **execute**: When multiple result sets are expected as the output of the call.

Let's take a look at the sample code snippet in *Listing 4.12* using a `CallableStatement` object and the `executeUpdate` method. Assume the `view_salary_increase` stored procedure is defined as:

```
view_salary_increase (IN p_empno varchar(6),
                    INOUT p_increase int,
                    OUT p_firstname)
```

where the stored procedure returns as the `OUT` parameter the first name of an employee for a given employee number.

```
(1) CallableStatement cstmt;
(2) cstmt = con.prepareCall("call view_salary_increase(?,?,?)");
(3) cstmt.setString(1,"000010");
(4) cstmt.setInt(2,10000000);
(5) cstmt.registerOutParameter(3, Types.VARCHAR);
(6) cstmt.executeUpdate();
(7) System.out.println(cstmt.getString(3) +
    " would receive and increase of " +
    cstmt.getInt(2));
(8) cstmt.close();
```

#### Listing 4.12 - CallableStatement using executeUpdate: Not returning a resultset

In the above listing:

- (1) The `CallableStatement` object `cstmt` is declared.
- (2) The `CallableStatement` object is created from the connection's method `prepareCall` and is assigned to `cstmt`
- (3) Input parameters to the stored procedure (`IN` or `INOUT`) must be set before executing the call
- (4) Same as 3, this parameter is an `INOUT` parameter.
- (5) Output parameters to the stored procedure must be registered to indicate the type. For `INOUT` parameter, if you `setXXX` first, it'd be optional to register it.
- (6) In this case the `executeUpdate` method is used because no result set is returned by the stored procedure.
- (7) Use the `getXXX` methods to retrieve the information from the stored procedure parameters (`OUT` and `INOUT`)
- (8) Closes the `CallableStatement` object `cstmt`

In this next example shown by the code snippet in *Listing 4.13*, a `CallableStatement` object is used to call the stored procedure `high_paid_employees` defined as:

```
high_paid_employees (IN p_salary INT))
```

where the stored procedure returns one result set with two columns: the first name and last name of employees with a salary greater than `p_salary`.

```
CallableStatement cstmt;
(1) cstmt = con.prepareCall("call high_paid_employees(?)");
(2) cstmt.setInt(1,80000);
(3) ResultSet rs = cstmt.executeQuery();
    System.out.println("High-paid employees list\n" +
        "-----");
(4) while ( rs.next() ) {
    System.out.println( rs.getString(1) + " " +
        rs.getString(2) );
    }
    rs.close();
    cstmt.close();
```

#### Listing 4.13 - CallableStatement object returning one result set

In the above listing:

- (1) The `CallableStatement` object `cstmt` is created from the connection's method `prepareCall` to invoke the `high_paid_employees` stored procedure.
- (2) Input parameter to the stored procedure is set before executing the call. There are no output parameters to register.
- (3) The `executeQuery` method issued since the stored procedure is returning one result set.
- (4) We loop through the result set, and use getter methods (`getXXX`) for the columns returned in the result set.

### 4.3.3 Receiving results

This section describes the third step when working with JDBC programs. It describes how to receive results, mainly `ResultSet` objects.

#### 4.3.3.1 ResultSet

A `ResultSet` object is returned by the `executeQuery` method of `Statement`, `PreparedStatement`, and `CallableStatement` objects. The `ResultSet` object maintains a cursor to the current row of a result set. This cursor can be moved to the next row by using the `next()` method of this object. A cursor by default can only be moved forward and is read-only; however, you could define a cursor to be scrollable or updatable.

The column value for the current row can be fetched by calling getter methods of the `ResultSet` object. When all the rows are fetched, the `next` method returns an exception.

Table 4.5 lists three properties for the `ResultSet` that can be set while creating the `Statement` object.

Property	Description
<code>resultSetType</code>	Defines the scrollability of the cursor
<code>resultSetConcurrency</code>	Defines the updatability of the cursor
<code>resultSetHoldability</code>	Indicates that the result set will remain open even after the statement is closed

**Table 4.5 - Properties for resultSet**

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. Listing 4.14 below provides an example of working with result sets taken from the IBM redbook mentioned at the beginning of section 4.3.

```
(1) Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                         ResultSet.CONCUR_UPDATABLE);
(2) ResultSet rs=stmt.executeQuery("Select POID,Status from
                                   purchaseorder");
(3) 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+".....");
(4)      rs.updateString(2,"Shipped");
(5)      rs.updateRow();
    }
}
(6) 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 );
}

```

#### Listing 4.14 - An example of using a ResultSet object

In the above listing:

- (1) This line is used to create the `statement` object with properties for the cursor to be scrollable (and sensitive to changes made by others) and updatable.
- (2) The `ResultSet` object is created
- (3) Looping through the `resultset`
- (4) For the condition where `id = 5003` and it is `UNSHIPPED`, update the String (column 2) for that row of the cursor to status of "Shipped". If there were more columns you could update the other columns too using updater methods (`updateXXX`)
- (5) Update the row (If you used `rs.insertRow()`, it'd insert a new row at that position)
- (6) Move the cursor back to the front of this `ResultSet` object, just before the first row.

### 4.3.4 Handling SQL errors and warnings

Just like any Java program, in JDBC, exception handling is done using the try-catch block. A DB2 application throws a `SQLException` whenever it encounters a SQL error or a `SQLWarning` whenever it encounters a SQL warning when executing SQL statements.

#### 4.3.4.1 SQLExceptions

An object of `SQLException` is created and thrown whenever an error occurs while accessing the database. The `SQLException` object provides the information listed in *Table 4.6*

SQLException information	Description	Method to retrieve this information
Message	Textual representation of the error code	<code>getMessage</code> method
SQLState	The SQLState string	<code>getSQLState</code> method.
ErrorCode	An integer value and indicates the error	<code>getErrorCode</code> method



	which caused the exception to be thrown.	
--	--	--

**Table 4.6 - SQLException information**

Apart from the above information, the 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.

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. *Listing 4.15* shows how to handle the `SQLException` in the try-catch block.

```
try {
// code which can throw SQLException go here
} catch (SQLException sqle)
{
    System.out.println("Rollback the transaction and quit the program");
    System.out.println();
    try { con.rollback(); }
        catch (Exception e) {}
    System.exit(1);
}
```

**Listing 4.15 - Handling a SQLException**

#### 4.3.4.2 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. *Listing 4.16* provides an example of working with `SQLWarning`.

```
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();
}
```

**Listing 4.16 - Handling a SQLWarning**

### 4.3.5 Closing the connection

We include this section for completeness of the steps to develop a JDBC program. However, we have already shown in earlier examples how to close a connection using the `close` method of a connection. Earlier in Listing 4.4, we also illustrated how to close `Statement` and `ResultSet` objects, also using their corresponding `close` methods.

### 4.3.6 Working with XML

Working with pureXML in Java applications is fairly easy. Simply work with SQL/XML and XQuery statements the way you normally work with SQL statements. Treat XML as a string within the Java program. There are performance advantages as well. The JDBC program does not need to retrieve the entire XML document as a string or CLOB (if that was the way it was stored) and build at run time a DOM tree (if using DOM parser). The tree was already built when the XML document was inserted into the database. Let DB2 software ("DB2") retrieve what you need based on the SQL/XML or XQuery you passed to it.

#### 4.3.6.1 Inserting XML data

If you want to insert a XML document to the database using a Java program, you have two choices:

- Hardcode the XML in the Java program and insert it as a string
- Store the XML document in a file, and insert the file using the `setBinaryStream` method

For example, let's assume you have created a table called *CLIENTS* with the following DDL:

```
create table clients(
    id                int primary key not null,
    name              varchar(50),
    status            varchar(10),
    contactinfo       xml
```

)

Note that the last column *contactinfo* is defined as XML. *Figure 4.7* shows the Java program treats this column as a string.

```

public static void insertString(){
    try {
        // for simplicity, I've defined variables with input data
        int id = 1885;
        String name = "Amy Liu";
        String status = "Silver";
        String xml =
            "<?xml version='1.0'?>" +
            "<Client>" +
            "<Address>" +
            "<street>54 Moorpark Ave.</street>" +
            "<city>San Jose</city>" +
            "<state>CA</state>" +
            "<zip>95110</zip>" +
            "</Address>" +
            "<phone>" +
            "<work>4084630110</work>" +
            "<home>4081114444</home>" +
            "<cell>4082223333</cell>" +
            "</phone>" +
            "<fax>4087776688</fax>" +
            "<email>sailer555@yahoo.com</email>" +
            "</Client>";

        // get a connection
        Connection conn = Conn.getConn();

        // define string that will insert file without validation
        String query = "insert into clients (id, name, status, contactinfo) values (?, ?, ?, ?)";

        // prepare the statement
        PreparedStatement insertStmt = conn.prepareStatement(query);
        insertStmt.setInt(1, id);
        insertStmt.setString(2, name);
        insertStmt.setString(3, status);
        insertStmt.setString(4, xml);

        // execute the statement
        if (insertStmt.executeUpdate() != 1) {
            System.out.println("No record inserted.");
        }
        conn.close();
    } catch (Exception e) { . . . }
}

```

**Figure 4.7 - Inserting an XML document hardcoded in the Java program**

In the figure above, a variable *xml* is defined as a String. Then in the line `insertStmt.setString(4, xml)`, the fourth parameter marker which corresponds to the XML column *contactInfo* is set with the value of the variable *xml*.

In *Figure 4.8*, the XML document is not hardcoded in the Java program, but stored in a file called `client1885.xml`. Then this file is inserted into the database using the `setBinaryStream` method. DB2 will take care of the rest.

```

public static void insertFile(){
    try {
        // for simplicity, I've defined variables with input data
        int id = 1885;
        String name = "Amy Liu";
        String status = "Silver";
        String fn = "c:/XMLFiles/Client1885.xml"; // input file

        // get a connection
        Connection conn = Conn.getConn();

        // define string that will insert file without validation
        String query = "insert into clients (id, name, status, contactinfo) values (?, ?, ?, ?)";

        // prepare the statement
        PreparedStatement insertStmt = conn.prepareStatement(query);
        insertStmt.setInt(1, id);
        insertStmt.setString(2, name);
        insertStmt.setString(3, status);
        File file = new File(fn);
        insertStmt.setBinaryStream(4, new FileInputStream(file), (int)file.length());

        // execute the statement
        if (insertStmt.executeUpdate() != 1) {
            System.out.println("No record inserted.");
        }
        conn.close();
    } catch (Exception e) { . . . }
}

```

Figure 4.8 - Inserting an XML document stored in a file

#### 4.3.6.2 Retrieving XML documents with SQL/XML and XQuery

Everything that was learned in *Chapter 15, DB2 pureXML*, can be applied to this chapter. *Figure 4.9* shows an example where an SQL/XML query is invoked in the Java program. A parameter marker is used, and for the second column of the result set returned (the email), it is retrieved as a string. The email as you can tell from the query is an element of the XML document stored in column *contactinfo*.

```

String status = "Silver";
try{
    // get a database connection
    // define, prepare, and execute a query that includes
    // (1) a path expression that will return an XML element and
    // (2) a parameter marker for a relational column value
    String query = "SELECT name, xmlquery('$c/Client/email[1]' " +
        " passing contactinfo as 'c' " +
        " from clients where status = ?)";
    PreparedStatement selectStmt = conn.prepareStatement(query);
    selectStmt.setString(1, status);
    ResultSet rs = selectStmt.executeQuery();

    // iterate over and print the results
    while(rs.next() ){
        System.out.println("Name: " + rs.getString(1) +
            " Email: " + rs.getString(2));
    }
    // release resources
} catch (Exception e) { . . . }

```

Figure 4.9 - Using SQL/XML in a Java program

Figure 4.10 is an example of using XQuery in a Java program. Again, there is nothing special about working with SQL/XML or XQuery in a Java program.

```
try{
    // get a database connection
    Connection conn = Conn.getConn();

    // define, prepare, and execute an XQuery (without SQL).
    // note that we must hard-code query predicate values.
    String query = "xquery for $y in db2-fn:xmlcolumn" +
        "('CLIENTS.CONTACTINFO')/Client " +
        "where $y/Address/city=\"San Jose\" and $y/Address/state=\"CA\" " +
        "return <emailList> { $y/email } </emailList>";
    PreparedStatement selectStmt = conn.prepareStatement(query);
    ResultSet rs = selectStmt.executeQuery();

    // iterate over all items in the sequence and print results.
    while(rs.next() ){
        System.out.println(rs.getString(1));
    }

    // release all resources
    // catch and handle any exceptions
    . . .
}
```

Figure 4.10 - Using XQuery in a Java program

**Note:**

A more complete explanation of JDBC including topics about database metadata, batch updates, transactions, savepoints, handling large objects, and more can be found in the IBM redbook *DB2 Express-C: The Developer Handbook for XML, PHP, C/C++, Java, and .NET (SG24-7301-00)*. See the Resources section in this book for more information

## 4.4 SQLJ Programming

SQLJ programming is a standard for embedding SQL statements into Java programs. All SQL statements are run statically using **contexts**. A context gives you information that helps interpret where the SQL statement is executed. There are different types of contexts:

- Connection context. This is equivalent to the **Connection** object in JDBC. A default connection context is used when no connection context is specified.
- Execution context. This is required to get the information regarding the SQL statement before and after executing the statement.

### 4.4.1 SQLJ Syntax

When working with SQLJ, there is different syntax that can be used that can help a precompiler identify the statements to translate from other statements in the embedded SQL Java program. There are different types of syntax, but they all start with “#sql” and use curly brackets as delimiters as shown below:

- #sql [connection-context] { sql statement }

- #sql [connection-context, execution context] { sql statement }
- #sql { sql statement }
- #sql [execution context] { sql statement }

Host variables can be identified by a colon as in the example below:

```
#sql {SELECT EMPNO FROM EMP WHERE WORKDEPT = :dept};
```

#### 4.4.2 Connection contexts

To work with SQL in an SQLJ program, you need to first establish a database connection. A connection context is used for that purpose. There are different ways to work with connection contexts as shown in *Listing 4.17* and *Listing 4.18* below.

```
(1) #sql context ctx; // This should be outside the class
(2) Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
(3) ctx ctx1 = new ctx("jdbc:db2:sample",false);
(4) #sql [ctx1] { DELETE FROM dept };
```

##### Listing 4.17 - Working with a Connection context

In the above listing:

- (1) Declare a class for the connection context using the syntax:

```
#sql context <context-class-name>
```

- (2) Load the JDBC driver; similar to JDBC programs.
- (3) Invoke the constructor of the `context` class.
- (4) An SQL statement (`DELETE`) is executed under the connection context "`ctx1`"

This other example in *Listing 4.18* is similar but it combines using JDBC connection objects with SQLJ.

```
#sql context ctx; // This should be outside the class
Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
(1) Connection con=DriverManager.getConnection();
(2) ctx ctx1 = new ctx(con);
#sql [ctx1] { DELETE FROM dept };
```

##### Listing 4.18 - Connection context from Connection object example:

In the above listing:

- (1) Using a `Connection` object
- (2) Invoke the constructor of the `context` class.

In addition, you can also create a default context for the connection. This means that later on there will not be a need to specify a context when performing SQL operations, as the context to use will be the one specified as the default one. The syntax would be of this form:

```
#sql { sql statement }
```

*Listing 4.19* provides an example.

```
Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
Connection con = DriverManager.getConnection();
(1) DefaultContext ctx1 = new DefaultContext(con);
(2) DefaultContext.setDefaultContext(ctx1);
(3) #sql { DELETE FROM dept };
```

#### **Listing 4.19 - Connection with default context example**

In the above listing:

- (1) A `DefaultContext` is created.
- (2) Set the default context to be `ctx1`
- (3) Specify the SQL to execute, in this case it is a `DELETE` statement. Note that there is no need to put the context name in the syntax. The default context `ctx1` will be used.

*Listing 4.20* below provides an example of a complete SQLJ program using a default context.

```
import java.sql.*;
(1) import sqlj.runtime.*;          // SQLJ runtime support
(2) import sqlj.runtime.ref.*;     // SQLJ runtime support

class myprg3 {
public static void main(String argv[]) {
    try {
        Connection con = null;
        Class.forName("com.ibm.db2.jcc.DB2Driver");
        String url = "jdbc:db2://127.0.0.1:50000/SAMPLE";
        if (argv.length == 2) {
            String userID = argv[0];
            String passwd = argv[1];
            con = DriverManager.getConnection(url,userID,passwd);
        }
    }
    else { throw new Exception
        ("\n Usage: java myprg3 userID password\n"); }
(3)         DefaultContext ctx = new DefaultContext(con);
(4)         DefaultContext.setDefaultContext(ctx);
```

```
(5)         if( ctx != null )
(6)         { ctx.close();}
           } catch (Exception e) { }
        } }
```

#### Listing 4.20 - A complete SQLJ program

In the above listing:

- 1 and 2: These packages need to be imported to provide for SQLJ runtime support
3. Creating a default context
4. Setting the default context to use
5. If the context is not closed
6. Close the context / disconnect

#### 4.4.3 Execution contexts

An execution context monitors and controls a SQL statement while executing; it is equivalent to the `Statement` interface in JDBC and is created within a connection context object.

To create an `ExecutionContext` object use the `getExecutionContext` method of the connection context. Some `ExecutionContext` methods work before an SQL statement is executed while others apply only after execution.

*Listing 4.21* provides an example.

```
#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);
(1) ExecutionContext exectx1 = ctx1.getExecutionContext();
(2) #sql[ctx1,exectx1] = { DELETE FROM purchaseorder WHERE
                        status='UnShipped' }
(3) int i = exectx1.getUpdateCount();
```

#### Listing 4.21 - An example using an execution context

In the above listing:

- (1) An execution context is created with the `getExecutionContext` method of the connection context object.
- (2) Specifying the SQL to run associated to connection context `ctx1`, and execution context `exectx1`.
- (3) Invoking the `getUpdateCount` method of the execution context object to obtain the number of records deleted or updated.



#### 4.4.4 Iterators

*Iterators* are equivalent to a JDBC result set. There are two types of iterators:

- **Named iterators:** Identify a row by the name of the column in the result set. While defining the named iterator, specify the name of the columns and their data types
- **Position iterators:** Identify a row by its position in the result set. While defining the position iterator, specify only the data types of the columns.

*Listing 4.22* provides an example of a named iterator.

```
(1) #sql iterator namediterator (int poid, String status)
(2) namediterator iterator1;
(3) #sql [ctx1] iterator1 = { select poid,status from purchaseorder };
(4) while(iterator1.next()) {
(5)   System.out.println("poid: " + iterator1.poid() + "Status: "+
      iterator1.status());
      }
(6) iterator1.close();
```

#### Listing 4.22 - An example of a named iterator

In the above listing:

- (1) A named iterator is declared with two columns. Note that the column names *poid* and *status* are explicitly mentioned. This is why this is a named iterator.
- (2) A named iterator *iterator1* is declared
- (3) In connection context *ctx1*, the *iterator1* is defined with a "select poid, status from purchaseorder"
- (4) Looping through the iterator
- (5) Printing the *poid* and *status* of each row retrieved. Note the syntax: "iterator1.poid()", and "iterator1.status()".
- (6) Closing *iterator1*.

*Listing 4.23* provides an example of a position iterator.

```
(1) #sql iterator positionedIterator (int, String);
(2) String status = null;
(3) int poid = 0;
(4) positionedIterator iterator1;
(5) #sql [ctx1] iterator1={ select poid, status from purchaseorder };
(6) #sql { fetch :iterator1 into :poid, :status };
(7) while(!iterator1.endFetch()) {
(8)   System.out.println("poid: " + poid + "Status: "+ status);
```

```
(9) #sql { fetch :iterator1 into :poid, :status };
}
```

#### Listing 4.23 - An example of a position iterator

In the above listing:

- (1) A position iterator is declared with two columns. Note that the column does not have names, just the data type. This is why this is a position iterator
- (2) A variable *status* is declared. This is like a host variable that are used to receive the values from the iterator
- (3) Similar to (2) for variable *poid*.
- (4) A positioned iterator *iterator1* is created.
- (5) In connection context *ctx1*, the *iterator1* is defined with a “select *poid*, *status* from purchaseorder”
- (6) Before starting the loop, we fetch the first record into the host variables *poid* and *status*. This is because in (7) we are not using `next()`, but `endFetch()` so the while loop condition would end differently if you use `next()` vs. `endFetch()`.
- (7) Looping through the iterator
- (8) Printing the *poid* and *status* of each row retrieved.
- (9) Fetching the next record in the iterator.

Named or position iterators can be updatable and scrollable. By default, iterators in SQLJ are read-only and can only move forward. To define a scrollable iterator, you need to implement `sqlj.runtime.Scrollable` while defining the iterator.

To define an updatable cursor, you need to implement `sqlj.runtime.ForUpdate` while defining the iterator. When defining an updatable iterator, you also need to specify the columns you would like to update.

This is similar as in JDBC. *Listing 4.24* provides an example of an updatable iterator.

```
(1) #sql public iterator namediterator implements sqlj.runtime.ForUpdate
    with (updateColumns="STATUS") (int poid, String status);
(2) namediterator iterator1;
(3) #sql [ctx1] iterator1={ select poid,status from purchaseorder };
(4) while(iterator1.next()) {
(5) System.out.println("before update poid: " + iterator1.poid() +
    "Status: " + iterator1.status());
(6) if(iterator1.status().toUpperCase().compareTo("UNSHIPPED")==0){
    #sql [ctx1] {update purchaseorder set status=
        'shipped' where current of :iterator1 };
    }
(7) #sql [ctx1] {commit};
```

**Listing 4.24 - An example of an updatable iterator**

In the above listing:

- (1) A named iterator is declared and it's defined as updatable iterator because it implements `sqlj.runtime.ForUpdate` and note it also indicates which column it will be updating in: `with (updateColumns="STATUS")`
- (2) The named iterator `iterator1` is declared
- (3) `iterator1` is defined as the result of the "select poid,status from purchaseorder"
- (4) Looping through the iterator
- (5) Printing each row of the iterator
- (6) Testing if the value of the `status` column is "UNSHIPPED". If it is, the value is changed to 'shipped'.
- (7) Committing the changes.

**4.4.5 Working with JDBC and SQLJ combined**

JDBC and SQLJ can be used together in a single application. For example, a JDBC connection object can be retrieved from a `ConnectionContext` object using its `getConnection` method and vice versa.

An iterator in SQLJ and a JDBC `ResultSet` can be retrieved from each other as shown below:

- Iterator from result set: `#sql iterator = {CAST :result-set }`
- Result set from an iterator: Use the iterator's `getResultSet` method.

*Listing 4.25* provides an example of working with JDBC and SQLJ combined in one program.

```
#sql public iterator positionIterator (int, String);
Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
Connection con=DriverManager.getConnection(url);
con.setAutoCommit(false);
(1) ctx ctx1=new ctx(con);
    positionIterator iterator;
    Statement stmt = con.createStatement();
    ResultSet rs=stmt.executeQuery("select poid, status from purchaseorder");
(2) #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();

```

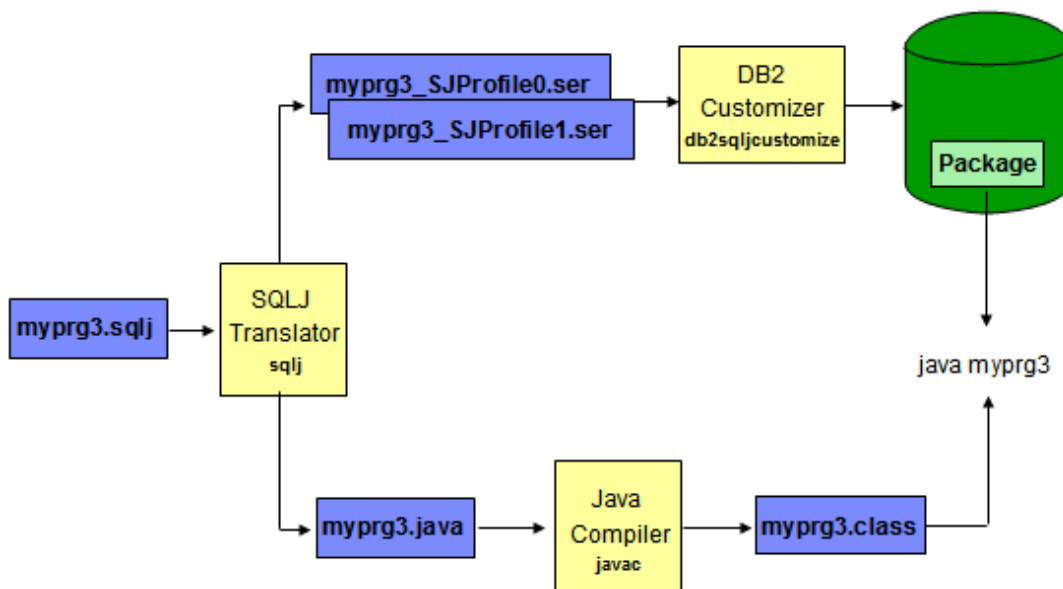
#### Listing 4.25 - Working with JDBC and SQLJ combined in one program

In the above listing:

- (1) The connection object is used to create an SQLJ connection context
- (2) This shows how an SQLJ iterator can be obtaining the results coming from a JDBC result set

#### 4.4.6 Preparing an SQLJ program

Preparing an SQLJ program is similar to the process followed for an embedded SQL program where you have to precompile and bind the program. In this case, the SQLJ program needs to be translated and customized. *Figure 4.11* shows the process to prepare the program `myprg3.sqlj`.



**Figure 4.11 - Steps to prepare a SQLJ program**

In the figure, the SQLJ program `myprg3.sqlj` goes through the **SQLJ Translator** using the `sqlj` command. The SQLJ Translator inspects the file looking for lines starting with `#sql1`, and replaces those lines with generated code that includes the SQLJ runtime classes, creating a file with `.java` extension. It then compiles this file using the Java compiler (`javac`) to create a `.class` file. This is shown at the bottom of the figure where the `myprg3.java` and the `myprg3.class` files are created.

The SQLJ translator will also create, as shown at the top of the figure, serialized profile files for each connection context class that is used in an SQLJ executable clause. In this example it shows two serialized profile files `myprg3_SJProfile0.ser` and

myprg3\_SJProfile1.ser. These files hold information about the embedded SQL. From these files a myprg3\_SJProfileKeys.class file (not shown in the figure) is created after javac is invoked. Using the DB2 customizer with the db2sqljcustomize command, a package (compiled SQL) is created in the database.

When you want to execute the program, simply run it using the command java myprg3 which will use the myprg3.class file and the corresponding package in the database.

**Note:**

The program myprg3.sqlj is included in the zip file Exercise\_Files\_DB2\_Application\_Development.zip accompanying this book. You can review, test, and prepare this file.

Below is an example of actually executing all of these commands and the corresponding output. These are the steps to follow:

**(1) Running the SQLJ Translator (sqlj command)**

```
D:\>sqlj myprg3.sqlj
```

The following files are created:

```
myprg3.java, myprg3.class,  
myprg3_SJProfile0.ser,  
myprg3_SJProfileKeys.class
```

**(2) Run the DB2 Customizer (db2sqljcustomize command) for every .ser file. The DB2 Customizer can take several parameters. It's basically a Java program that needs to connect to a database since that's where it will store the package. For example:**

```
D:\>db2sqljcustomize -url jdbc:db2://localhost:50000/sample -user  
rfchong -password mypasswd myprg3_SJProfile0.ser
```

Where:

- -url jdbc:db2://localhost:50000/sample is the URL needed to connect to the database
- -user rfchong is user ID to connect to the database
- -password mypasswd is the password to connect to the database
- myprg3\_SJProfile0.ser is the file name to customize to create the package in the database

The output would look as shown in *Listing 4.26* below.

```
[jcc][sqlj]
[jcc][sqlj] Begin Customization
[jcc][sqlj] Loading profile: myprg3_SJProfile0
[jcc][sqlj] Customization complete for profile myprg3_SJProfile0.ser
[jcc][sqlj] Begin Bind
[jcc][sqlj] Loading profile: myprg3_SJProfile0
[jcc][sqlj] Driver defaults(user may override): BLOCKING ALL
VALIDATE BIND STATICREADONLY YES
[jcc][sqlj] Fixed driver options: DATETIME ISO DYNAMICRULES BIND
[jcc][sqlj] Binding package MYPRG301 at isolation level UR
[jcc][sqlj] Binding package MYPRG302 at isolation level CS
[jcc][sqlj] Binding package MYPRG303 at isolation level RS
[jcc][sqlj] Binding package MYPRG304 at isolation level RR
[jcc][sqlj] Bind complete for myprg3_SJProfile0
```

**Listing 4.26 - Output of running the DB2 customizer**

No output files will be created, but a package with the embedded SQL statements and access plan would be stored in the database.

(3) Run the program. The package previously created is then used at runtime.

```
D:\>java myprg3 <userid> <password>
Successful connection to the database!
Successful retrieval of record. Column 'IBMREQD' has a value of 'Y'
Successful Disconnection from database
End of Program
```

*Figure 4.12* below shows all the output of preparing the program myprg3.sqlj

```

c:\ DB2 CLP - DB2COPY1
D:\Temp>dir
Volume in drive D is Local Disk
Volume Serial Number is 7C98-5C4B

Directory of D:\Temp

07/30/2009  08:57 AM    <DIR>          .
07/30/2009  08:57 AM    <DIR>          ..
07/25/2009  03:36 PM                4,567 myprg3.sqlj
               1 File(s)                4,567 bytes
               2 Dir(s)      22,044,499,968 bytes free

D:\Temp>sqlj myprg3.sqlj

D:\Temp>dir
Volume in drive D is Local Disk
Volume Serial Number is 7C98-5C4B

Directory of D:\Temp

07/30/2009  08:58 AM    <DIR>          .
07/30/2009  08:58 AM    <DIR>          ..
07/30/2009  08:58 AM                3,140 myprg3.class
07/30/2009  08:57 AM                6,835 myprg3.java
07/25/2009  03:36 PM                4,567 myprg3.sqlj
07/30/2009  08:57 AM                1,553 myprg3_SJProfile0.ser
07/30/2009  08:58 AM                1,107 myprg3_SJProfileKeys.class
               5 File(s)                17,202 bytes
               2 Dir(s)      22,044,479,488 bytes free

D:\Temp>db2sqljcustomize -url jdbc:db2://localhost:50000/sample -user arfchong -
password mypasswd myprg3_SJProfile0.ser
[jcc]sqlj]
[jcc]sqlj] Begin Customization
[jcc]sqlj] Loading profile: myprg3_SJProfile0
[jcc]sqlj] Customization complete for profile myprg3_SJProfile0.ser
[jcc]sqlj] Begin Bind
[jcc]sqlj] Loading profile: myprg3_SJProfile0
[jcc]sqlj] Driver defaults(user may override): BLOCKING ALL VALIDATE BIND STATI
CREADONLY YES
[jcc]sqlj] Fixed driver options: DATETIME ISO DYNAMICRULES BIND
[jcc]sqlj] Binding package MYPRG301 at isolation level UR
[jcc]sqlj] Binding package MYPRG302 at isolation level CS
[jcc]sqlj] Binding package MYPRG303 at isolation level RS
[jcc]sqlj] Binding package MYPRG304 at isolation level RR
[jcc]sqlj] Bind complete for myprg3_SJProfile0

D:\Temp>java myprg3 arfchong mypasswd
Successful connection to the database!
Successful retrieval of record. Column 'IBMREQD' has a value of 'Y'
Successful Disconnection from database
End of Program

D:\Temp>

```

Figure 4.12 - Preparing the SQLJ program myprg3.sqlj

## 4.5 pureQuery

At this point you should have a basic understanding about JDBC and SQLJ. Traditional JDBC and SQLJ programming, as you may have noticed, requires some tedious programming. This is one of the reasons why **object relational mapping (ORM) frameworks** such as Hibernate are popular. They provide a data access abstraction layer that facilitates the mapping between your object-oriented code and your relational database model. However, ORM frameworks tend to generate the SQL required behind the scenes for you, and this generated SQL may not be optimal. Moreover, diagnosing

performance issues and tuning become more complex because the developers no longer control what SQL is sent to the database.

In answer to these challenges, IBM developed **pureQuery**. You can think of pureQuery as a thin layer of APIs that sits on top of JDBC. It facilitates the mapping of object-oriented code with the relational model as ORM frameworks do; but it also gives you the flexibility to work with your SQL. Therefore, pureQuery programming model and tools, help with Java data access, improves performance by generating code that uses best practice approaches such as using JDBC-like batch updates, and helps with problem determination.

In addition to APIs, pureQuery also provides the following:

- A runtime, which provides optimized and secure database access
- An Eclipse-based integrated database development environment for enhancing development productivity
- Monitoring services, to provide developers and DBAs with previously unknown insights about performance of Java database applications.

pureQuery also allows you to generate static SQL without changing any code. This unlocks the advantages of static SQL without any effort.

The tools for pureQuery are included in the product **Optim Development Studio**. The APIs and runtime are available with the product Optim™ pureQuery Runtime and is also packaged at no extra charge for development use on the developer's machine with Optim Development Studio. Monitoring services are delivered in Optim Development Studio and to a greater degree with the IBM DB2 Performance Expert and Extended Insight Feature.

IBM Optim™ portfolio of products, provide an integrated, modular environment to design, develop, deploy, operate, optimize and govern enterprise data throughout its lifecycle. This is known as **Optim Integrated Data Management**.

**Note:**

More information about pureQuery and Optim Integrated Data Management can be found in the free e-books *Getting Started with pureQuery*, and *Getting Started with IBM Data Studio for DB2* respectively. Both eBooks are part of this book series.

## 4.6 Exercises

In this exercise you will practice writing small JDBC and SQLJ programs.

### Procedure

1. In *Listing 4.12* we illustrated an example where a `CallableStatement` object was used to return one result set. Create a JDBC program with this specifications:

- Program name: `mylab4.java`
- The program should be executed with using this syntax:



- ```
java mylab4 <userid> <password> <bonus>
```
- Connect to the **SAMPLE** database using the type 4 driver
  - Call the "*high\_bonus*" stored procedure which returns 2 result sets
  - Ensure you first create the *high\_bonus* stored procedure as shown below:

```
CREATE PROCEDURE HIGH_BONUS (IN p_bonus INT)
  DYNAMIC RESULT SETS 2
-----
-- This procedure lists the first name, last name, bonus, and
-- education level (edlevel) of the employees with a bonus amount
-- larger than the amount specified by the parameter p_bonus.
-- This information is in the EMPLOYEE table.
-- The procedure also returns another result set with the names
-- of all departments from the DEPARTMENT table.
-----
P1: BEGIN
  -- Declare cursor1
  DECLARE cursor1 CURSOR WITH RETURN FOR
    SELECT firstnme, lastname, bonus, edlevel
       FROM employee
       WHERE bonus > p_bonus;

  -- Declare cursor2
  DECLARE cursor2 CURSOR WITH RETURN FOR
    SELECT deptname
       FROM department;

  -- Cursor left open for client application
  OPEN cursor1;
  OPEN cursor2;
END P1
```

The program `mylab4.java` (solution) is included in the `Exercise_Files_DB2_Application_Development.zip` file that accompanies this book.

2. Look for the file `connectionContext.sqlj` in the `Exercise_Files_DB2_Application_Development.zip` file that accompanies this book. Prepare this SQLJ program and execute it.

## 4.7 Summary

In this chapter you learned the basics of developing Java applications working with DB2. JDBC, a standard API to access databases using dynamic SQL, was discussed first. The chapter showed you how to connect to a DB2 database and issue different kinds of JDBC statements. Then you were introduced to SQLJ. Though not that popular, SQLJ programming is more compact and better for performance as it uses static SQL embedded in Java programs. Finally, we briefly introduced you to IBM's pureQuery. pureQuery is IBM's answer to tedious JDBC programming and lack of flexibility of ORM frameworks.

## 4.8 Review questions

1. What is the difference between JDBC and SQLJ?
2. What JDBC types are supported with DB2 9.7?
3. What is the difference between `db2jcc.jar` and `db2jcc4.jar`?
4. What is an iterator?
5. What is a default context?
6. Which of the following statements is false?
  - A. `db2jcc.jar` includes a JDBC type 2 driver
  - B. `db2jcc4.jar` includes a JDBC type 2 driver
  - C. `db2jcc.jar` includes a JDBC type 4 driver
  - D. `db2jcc4.jar` includes a JDBC type 4 driver
  - E. None of the above
7. Which of the following statements is true?
  - A. pureQuery allows developers to run SQL as dynamic or static easily with no code change
  - B. pureQuery can be used with an ORM
  - C. pureQuery is included with some Optim products
  - D. All of the above
  - E. None of the above
8. Which of the following statements is true?
  - A. JDBC and SQLJ cannot be combined
  - B. JDBC and pureQuery cannot be combined
  - C. SQLJ and pureQuery cannot be combined
  - D. JDBC, SQLJ and pureQuery can be combined

- E. None of the above
9. Which of the following is not a valid connection object method?
- A. Statement
  - B. PreparedStatement
  - C. CallableStatement
  - D. ResultSetStatement
  - E. None of the above
10. Which of the following is not a valid SQLJ concept?
- A. Executive context
  - B. Connection context
  - C. Default context
  - D. Named iterator
  - E. Position iterator



# 5

## Chapter 5 – Application development with C/C++

This chapter discusses the different aspects of application development using C/C++ with DB2, from setting up the development environment to building and running the application. Though embedded SQL was discussed earlier in *Chapter 14, Introduction to DB2 Application Development*, we will provide a closer look at how embedded SQL programming works with a C/C++ program. The chapter also discusses how to develop an ODBC or CLI application. In this chapter you will learn about:

- Programming a C/C++ application with embedded SQL
- Building a C/C++ application with embedded SQL
- Programming a C/C++ application with CLI/ODBC
- Building a C/C++ application with CLI/ODBC

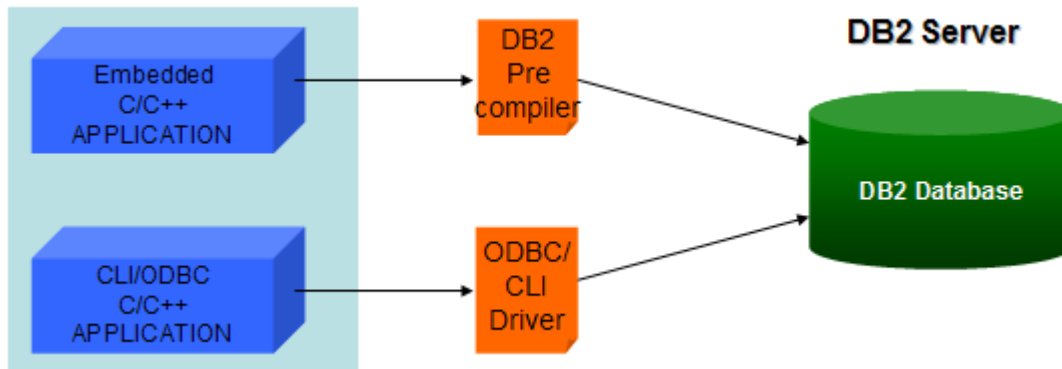
### 5.1 C/C++ DB2 applications: The big picture

When we talk about database applications, the first thing that comes to our mind is SQL. SQL is the language which is used to perform Data Definition Language (DDL), Data Control Language (DCL), and Data Manipulation Language (DML) operations on a relational database. SQL is not a general purpose programming language; you cannot perform user defined operations as SQL is a non-procedural language where the SQL statements are executed by the database manager. That's why database applications are mostly developed by combining the capabilities of SQL with a high level programming language such as C or C++. Developing an application in a high level programming language -- also known as the host language, and embedding SQL statements in that application is known as embedded SQL programming. *Figure 5.1* illustrates this concept. Embedded SQL applications are commonly developed using C or C++.



Figure 5.1 – Embedded C/C++ SQL application

A C/C++ application works in a client-server architecture where the C/C++ application works as the client and DB2 as the server. The client sends the request to the server and the server performs the operation. A C/C++ application can be an embedded SQL application or a CLI/ODBC application. *Figure 5.2* illustrates the client-server architecture where a C/C++ application interacts with a DB2 server.



**Figure 5.2 – C/C++ application accessing a DB2 server**

*Figure 5.2* shows how C/C++ applications are built and can access a DB2 database. Embedded SQL applications containing SQL statements must be precompiled by a DB2 precompiler. To facilitate this preprocessing, each SQL statement embedded in an application must be prefixed with the keywords EXEC SQL and terminated with a semicolon (;).

On the other hand CLI or ODBC is an alternative to embedded dynamic SQL, but unlike embedded SQL, it does not require host variables or a precompiler. Applications can be run against a variety of databases from different vendors which would provide their own ODBC drivers. There is no need for the application to be compiled against each of these databases. Processing of ODBC/CLI applications is handled by the ODBC/CLI driver. Applications use procedure calls at run time to connect to databases, issue SQL statements, and retrieve data and status information.

We will discuss each of the above methods in more detail in the coming sections.

## 5.2 Setting up the environment

Before you start building an embedded SQL C/C++ application or an ODBC/CLI application, you need to install a supported C/C++ compiler. For embedded SQL C/C++ programs, you also need a precompiler; fortunately, this is included with DB2.

### 5.2.1 Supported compilers

*Table 5.1* lists C/C++ compilers that are supported for DB2 database application development on both, 32-bit and 64-bit platforms.

| Operating System | Supported Compilers                                                                                                                                               |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| AIX®             | IBM XL C/C++ Enterprise Edition Version 7.0 for AIX<br>IBM XL C/C++ Enterprise Edition Version 8.0 for AIX<br>IBM XL C/C++ Enterprise Edition Version 9.0 for AIX |
| Linux            | GNU/Linux gcc versions 3.3 and 3.4<br>Intel C Compiler Version 9.1<br>Intel C Compiler Version 10.1                                                               |
| Windows®         | Intel Proton Compiler for Windows 32-bit applications, Version 9.0.021 or later<br>Microsoft® Visual C++ .NET or later                                            |

**Table 5.1 – Supported C/C++ compilers**

**Note:**

This list is not a complete list. For more details and the latest information about the supported C/C++ compiler versions, refer to the DB2 9.7 Information Center at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.gs.doc/doc/r0023434.html> and the DB2 Application Development Web site at: <http://www.ibm.com/software/data/db2/udb/ad/>

In this book we use the Microsoft Visual C++ compiler (Windows) and the GNU/Linux gcc compiler (Linux). Microsoft Visual C++ compiler is licensed software and comes with Microsoft Visual Studio. The Express version of Visual Studio can be downloaded from <http://www.microsoft.com/express/Downloads/>.

GNU Linux gcc compiler comes free with GNU operating systems, but can also be downloaded from <http://gcc.gnu.org/>

## 5.2.2 Setting up the C/C++ environment

To set up the C/C++ environment, follow these steps:

1. Verify a supported C/C++ compiler is installed

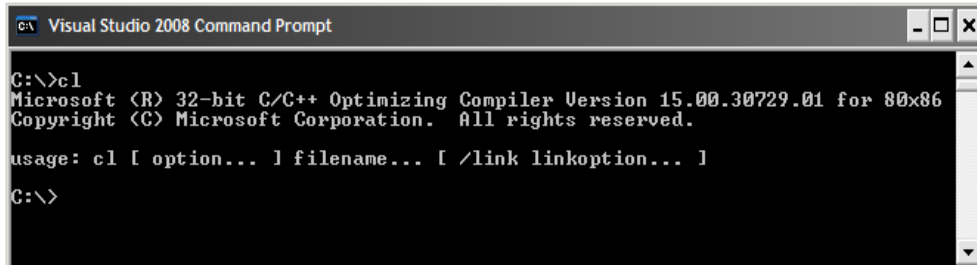
Make sure that a supported C/C++ compiler is installed on a DB2 Express-C supported platform. Check *Table 5.1* for a list of supported compiler versions.

On Linux, to check whether a C/C++ compiler is installed successfully or not, issue the command below at the Linux shell:

```
which gcc
```

If the compiler is installed, you should get `/usr/bin/gcc` on the screen.

On Windows, if using the Microsoft Visual C++ compiler, issue the `cl` command. *Figure 5.3* shows the output after running this command.



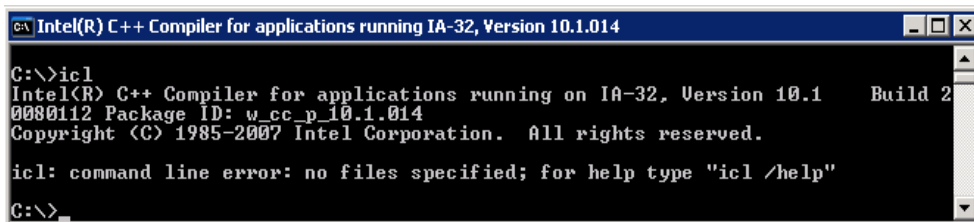
```
C:\>cl
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.01 for 80x86
Copyright (C) Microsoft Corporation. All rights reserved.

usage: cl [ option... ] filename... [ /link linkoption... ]

C:\>
```

**Figure 5.3 – Output of the `cl` command**

If using the Intel compiler, issue either the `icl` or `ec1` commands. *Figure 5.4* shows the output after running this command.



```
C:\>icl
Intel(R) C++ Compiler for applications running on IA-32, Version 10.1 Build 2
0080112 Package ID: w_cc_p_10.1.014
Copyright (C) 1985-2007 Intel Corporation. All rights reserved.

icl: command line error: no files specified; for help type "icl /help"

C:\>
```

**Figure 5.4 – Output of the `icl` command**

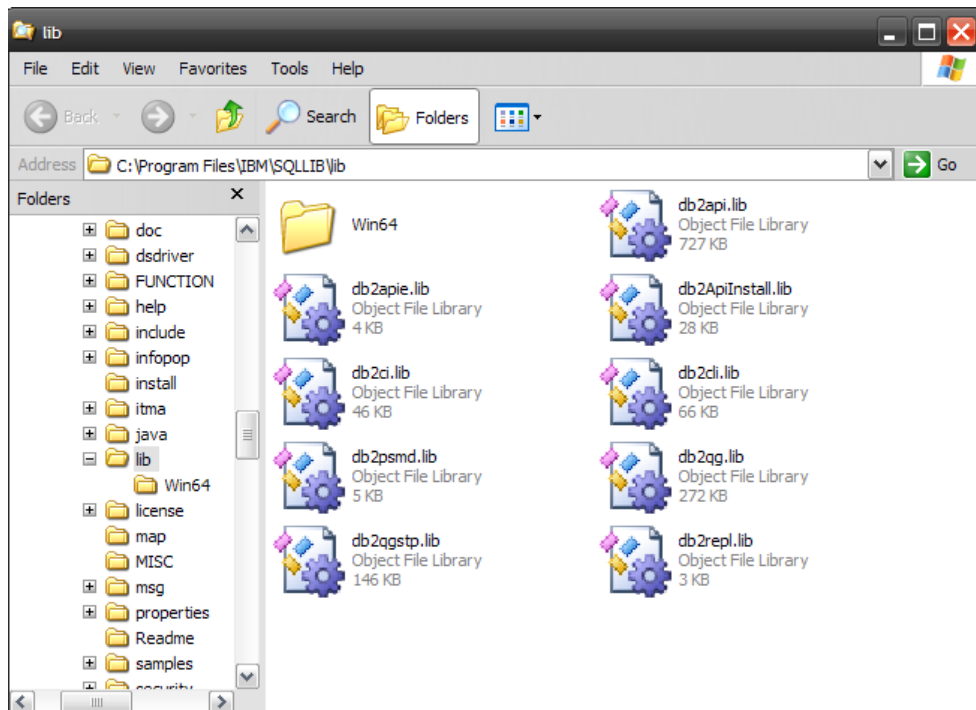
2. Verify your DB2 installation has the required libraries and header files

After installing DB2 verify that the necessary static and shared libraries and header files to develop C/C++ programs are present.

On Linux, this can be done by going to the DB2 install directory and checking for the `lib32` and `lib64` directory. For example if the DB2 install directory is in `$HOME`, then the 32-bit libraries location will be in `$HOME/sql1lib/lib32`, and the 64-bit libraries location will be in `$HOME/sql1lib/lib64`.

On Windows, the static libraries are located at `<DB2 install directory>\lib` *Figure 5.5*. shows the location of static libraries. Here the DB2 install directory is `C:\Program Files\SQLLIB`





**Figure 5.5 – Location of libraries and header files on a DB2 Windows installation**

### 3. Verify the Windows environment

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, follow one of these options:

- Modify the INCLUDE variable at a command prompt by running the command: `set INCLUDE=%DB2PATH%\include;%INCLUDE%` (This will work for your current Window)
- If you want this change to be permanent, locate the Windows environment variables by right-clicking on *My Computer* -> *Properties* -> *Environment Variables* and adding %DB2PATH%\include in the INCLUDE variable.

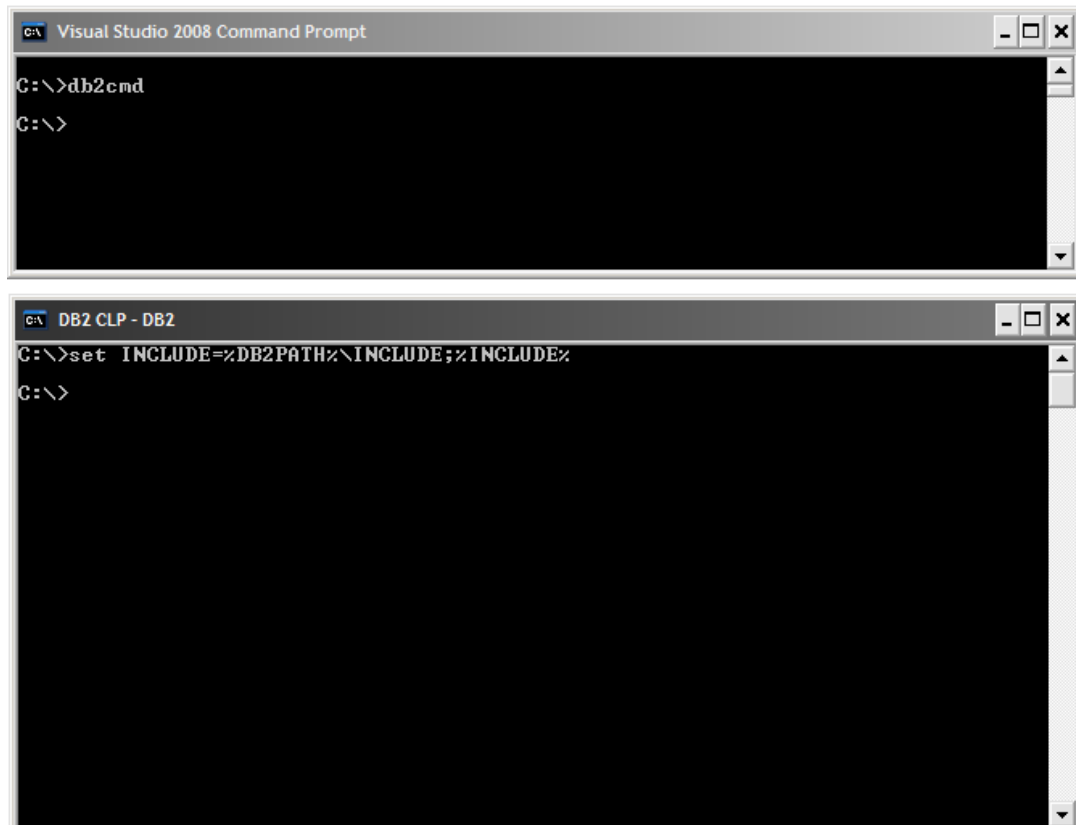
For development using Visual Studio, you must ensure that the INCLUDE environment variable contains %DB2PATH%\INCLUDE as the first directory. For this you need to update the environment for your compiler using these steps:

1. Open the shortcut to the Visual Studio Command Prompt by going to *Program Files* -> *Microsoft Visual Studio .NET* -> *Visual Studio .NET Tools* -> *Visual Studio .NET Command Prompt*
2. In the Visual Studio Command Prompt window, run `db2cmd.exe` to open the DB2 Command Window.

3. In the DB2 command window, set your `INCLUDE` path as follows:

```
set INCLUDE=%DB2PATH%\INCLUDE;%INCLUDE%
```

Figure 5.7 shows the above steps:-



**Figure 5.7 – Setup windows environment**

Now you can build your application on the DB2 CLP window which has the correct DB2 environment and C/C++ environment set up.

## 5.3 Developing a C/C++ application with embedded SQL

This section discusses the steps required to develop a C/C++ application with embedded SQL.

### 5.3.1 Source file extensions

The C/C++ source files for an embedded C/C++ SQL program need appropriate file extensions that can be recognized by the DB2 precompiler. *Table 5.2* lists the C/C++ source file extensions required by the precompiler.

| Platform | C source File | C++ source file |
|----------|---------------|-----------------|
| Linux    | .sqc          | .sqC            |
| Windows  | .sqc          | .sqx            |

**Table 5.2 – Supported source file extensions**

### 5.3.2 SQL data types in C/C++

To communicate between the application and the database, usage of correct data type mapping (between the C/C++ data type of the host variable and the SQL data type) is very important. When the precompiler finds the declaration of a host variable, it determines the appropriate SQL data type. *Table 5.3* lists some supported SQL data types in C/C++.

| SQL column type | C and C/C++ data type                                           | SQL column type description                    |
|-----------------|-----------------------------------------------------------------|------------------------------------------------|
| SMALLINT        | Short                                                           | 16 bit signed integer                          |
| INTEGER         | sqlint32                                                        | 32 bit signed integer                          |
| DOUBLE          | Double                                                          | Double-precision floating point                |
| CHAR(n)         | char[n+1] where n is large enough to hold the data<br>1<=n<=254 | Fixed-length, null-terminated character string |
| VARCHAR(n)      | char[n+1] where n is large enough to hold the data<br>1<=n<=254 | Null-terminated varying length string          |
| DATE            | char[11]                                                        | Null-terminated character form                 |
| TIME            | char[9]                                                         | Null-terminated character form                 |

**Table 5.3 – Supported SQL data types in C/C++**

**Note:**

This list is not a complete list. The full mapping list can be found at:

<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.db2.luw.apdv.embed.doc/doc/r0006090.html>

### 5.3.3 Steps to develop an embedded SQL C/C++ application

Developing a C/C++ application with embedded SQL involves the following steps:

1. Include required header files
2. Declare host variables
3. Connect to a database
4. Execute SQL statements
5. Handle SQL errors
6. Commit the transactions
7. Disconnect from the database

We will explore each step one by one. *Listing 5.1* illustrates a basic template you can use with the structure required for an embedded SQL application written in C.

```
(1) /* Include required header files

(2) /* Declaration of host variables */
EXEC SQL BEGIN DECLARE SECTION;
.....
.....
EXEC SQL END DECLARE SECTION;

/* Declaration of SQLCA structure */
EXEC SQL DECLARE SQLCA;

/* Declaration of main function */
void main()
{
  /* Connect to the database */
(3) EXEC SQL CONNECT TO <database name> ;

/* Error handling to check connection is successful*/
if (SQLCODE < 0)
{
  printf ("\n Error while connecting to database");
  printf ("\n Returned SQLCODE = ");
  exit;
}
else
{
  printf ("\n Connect to database successfully");
```

```
    }
    /* End of error handling */

(4) /* Execute SQL statements
EXEC SQL SELECT <col1>, <col2> INTO :var1 :var2
      FROM <table name> WHERE <condition> ;
/* Error handling to check SQL statement executed successfully */
(5) if (SQLCODE < 0)
    {
        printf ("\n Error while executing SQL statement");
        printf ("\n Returned SQLCODE = ");
        exit;
    }

(6) /* Commit the transaction */
EXEC SQL COMMIT;
/* Error handling to check SQL statement executed successfully */
if (SQLCODE < 0)
    {
        printf ("\n Error while Committing data");
        printf ("\n Returned SQLCODE = ");
        exit;
    }

(7) /* Disconnect from the database */
EXEC SQL DISCONNECT FROM <database name>;

/* Error handling to check whether disconnection is successful */
if (SQLCODE < 0)
    {
        printf ("\n Error while disconnecting from database");
        printf ("\n Returned SQLCODE = ");
        exit;
    }
else
    {
        printf ("\n Disconnect from database successfully");
    }
}
```

**Listing 5.1 – Embedded SQL C program template**

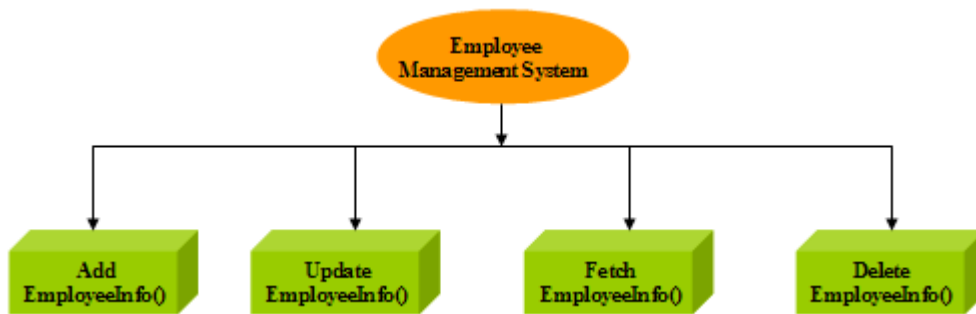
Each of the items numbered in *Listing 5.1* are discussed in detail in the following subsections.

**Note:**

The above template is in the file `template.sqc` which is included in `Exercise_Files_DB2_Application_Development.zip` file that accompanies this book.

**5.3.4 Sample embedded SQL C/C++ application**

To illustrate the different steps mentioned in the previous section, we will use a sample application. This application is for an employee management system where you can add, update, fetch and delete employee information from a table. *Figure 5.8* provides an overview of what the application does.



**Figure 5.8 – Sample embedded SQL C/C++ application functions**

*Table 5.4* describes the functions shown in *Figure 5.8*.

| Function              | Operation                                    |
|-----------------------|----------------------------------------------|
| AddEmployeeInfo( )    | INSERT new employee information in the table |
| UpdateEmployeeInfo( ) | UPDATE employee salary into the table        |
| FetchEmployeeInfo( )  | SELECT employee information from the table   |
| DeleteEmployeeInfo( ) | DELETE employee information from the table   |

**Table 5.4 – Functions for different operations**

**Note:**

All code snippets shown in this section are extracted from the program `embeddedC.sqc` which is included in the file `Exercise_Files_DB2_Application_Development.zip` accompanying this book.

**5.3.4.1 Include required header files**

The first step to create an embedded application is to include the required header files. `sqlca.h` is probably the most important one pertaining to SQL, and the rest would depend

on what your C application does. *Listing 5.2* lists the typical header files required for an embedded SQL C application.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlca.h>
```

**Listing 5.2 – Include required header files**

### 5.3.4.2 Declare Host variables

As discussed in *Chapter 1 - Introduction to DB2 Application Development*, an embedded SQL application communicates to the C/C++ host language with the help of host variables. In this second step you must declare the host variables at the top of the program in the DECLARE section. The host variable DECLARE SECTION starts with the EXEC SQL BEGIN keyword and ends with EXEC SQL END keyword as illustrated in *Listing 5.3*.

```
EXEC SQL BEGIN DECLARE SECTION;
sqlint32 hempno;          /* to store employee id */
char hfirstname[20];     /* to store employee first name */
char hlastname[20];     /* to store employee last name */
char hedlevel[4];       /* to store employee education level */
double hsalary ;        /* to store employee salary */
double hnewsalary;      /* to store employee new salary */
char hdynstmt[16384];    /* to store SQL statements*/
EXEC SQL END DECLARE SECTION;
```

**Listing 5.3 – Declare host variables**

### 5.3.4.3 Connect to the database

Before you can perform operations on the database, you need to establish a connection. Database connections can be established implicitly or explicitly. An implicit connection is a connection where the user ID is assumed to be the current user ID. Explicit connection is a connection, which requires a user ID and password to be specified in the connect statement. In C/C++ applications, an implicit database connection can be established by executing the following statement:

```
EXEC SQL CONNECT TO sample;
```

If you want to establish an explicit connection use the following statement:

```
EXEC SQL CONNECT TO sample USER administrator USING admin123;
```

*Listing 5.4* shows the connect statement with error handling

```
(1) EXEC SQL CONNECT TO sample;      /* Database name is SAMPLE*/
(2) if (SQLCODE < 0)
    {
        printf ("\n ERROR WHILE DATABASE CONNECTION\n");
        printf ("\n SQLCODE = %d", SQLCODE);
        rc = 1;
        exit(1);
    }
else
    {
        printf ("\n CONNECT TO DATABASE SUCCESSFULLY\n");
    }
}
```

#### Listing 5.4 – Connect to the database

Let's review each of the items shown in *Listing 5.4*:

1. This statement establishes the connection to the `sample` database.
2. This statement checks if the connection is successful or not. `SQLCODE` is a special variable in DB2 that is set after a statement is executed. When `SQLCODE` is less than zero, the statement completed with an error. If `SQLCODE` is equal to zero, the statement completed successfully; and if it was greater than zero, the statement completed with a warning.

*Figure 5.9* provides the output of above code snippet.

A screenshot of a terminal window titled "Select DB2 CLP - DB2 - embeddedC". The terminal shows the command prompt "C:\books\cc++>embeddedC" followed by the command "CONNECT TO DATABASE". The output of the command is "CONNECT TO SAMPLE DATABASE SUCCESSFULLY".

```
C:\books\cc++>embeddedC
CONNECT TO DATABASE
CONNECT TO SAMPLE DATABASE SUCCESSFULLY
```

#### Figure 5.9 – Output of the connect statement

If you include the `utilemb.h` header file, you can also use the `DbConn` utility function that comes with DB2 sample programs to connect to the database. *Listing 5.5* shows how to use `Dbconn` utility function.

```
(1) rc = DbConn(dbAlias, user, pswd);
(2) if (rc != 0)
    {
        printf("\n Error while connecting to database");
        return rc;
    }
}
```

#### Listing 5.5 – Connect to the database using `DbConn` utility function

Let's review each of the items shown in *Listing 5.5*:



1. This statement establishes the connection to the sample database using DbConn utility function. This function takes a userid, a password, and a database name as arguments.
2. This statement checks if the connection was successful or not.

**Note:**

The utility functions included with the DB2 sample programs for C are in the directory <DB2 install path>\sqllib\samples\c (on Windows) and <DB2 install path>/sqllib/samples/c (on Linux). For C++ look for the subdirectory "cpp" as in <DB2 install path>\sqllib\samples\cpp (on Windows)

**5.3.4.4 Execute SQL statements**

To execute SQL statements, EXEC SQL keywords are needed to indicate the beginning of a SQL statement and must be terminated by a semicolon (;). In this section we will see how to execute static and dynamic SQL statements and perform different operations like **INSERT**, **SELECT**, **UPDATE** and **DELETE**.

**5.3.4.4.1 Inserting data into a table**

Let's take a look at the function **AddEmployeeInfo()** illustrated in *Listing 5.6* below. This function is used to insert employee's information into the EMPLOYEE table.

```

int AddEmployeeInfo()
{
    int rc = 0;
    printf("\n=====");
    printf("\n ADD EMPLOYEE INFORMATION");
    printf("\n=====");
(1) EXEC SQL INSERT INTO employee(empno, firstnme, lastname, edlevel,
    salary)
    VALUES (50001, 'RAUL', 'CHONG', 21, 6000),
            (50002, 'JAMES', 'JI', 20, 5786),
            (50003, 'MIN', 'YAO', 20, 5876),
            (50004, 'IAN', 'HAKES', 19, 5489),
            (50005, 'VINEET', 'MISHRA', 19, 5600);
(2) if(SQLCODE < 0)
    {
        printf ("\n ERROR WHILE ADDING EMPLOYEE INFORMATION");
        printf ("\n RETURNED SQLCODE = %d", SQLCODE);
        rc = -1
    }
    else
    {
        printf("\n EMPLOYEE ADDED SUCESSFULLY ");
    }
}

```

```

EXEC SQL COMMIT;
}
return rc;
}

```

### Listing 5.6 – INSERT employee information

Let's review each of the items shown in *Listing 5.6.*:

1. This statement inserts the employee information into the employee table.
2. This statement checks whether the insert was successful.

*Figure 5.10* provides the output of the above code snippet.

```

=====
EMPLOYEE MANAGEMENT SYSTEM
=====
1. Add New Employee
2. Update Employee Information(Salary)
3. Fetch Employee Information
4. Delete Employee Information
5. Exit

Enter option: 1

=====
ADD EMPLOYEE INFORMATION
=====
EMPLOYEE ADDED SUCESSFULLY

```

**Figure 5.10 – Output of the INSERT operation**

#### 5.3.4.4.2 Retrieving data from a table

Retrieval of data is done by using a SELECT statement. If the SELECT statement will return a single row, use the INTO clause (SELECT INTO) so the results are stored directly into host variables specified.

If the SELECT returns more than one row, you must use a **cursor** to fetch them one at a time. A cursor is a named control structure used by an application program to point to a specific row within an ordered set of rows. The steps involved to work with a cursor are:

1. Declare the cursor using a DECLARE CURSOR statement.
2. Open the cursor using the OPEN statement.
3. Retrieve rows one at a time using the FETCH statement.
4. Terminate the cursor using the CLOSE statement.

Let's take a look at the function `FetchEmployeeInfo()` in *Listing 5.7* for an example of retrieving data. This function retrieves employees' information for those employees with and ID between 50001 and 50005 from the EMPLOYEE table.

```

int FetchEmployeeInfo()
{

```

```

int rc = 0;
(1) EXEC SQL DECLARE cur1 CURSOR FOR SELECT empno, firstnme, lastname,
    edlevel, salary FROM employee WHERE empno BETWEEN 50001 AND 50005;
    if(SQLCODE < 0)
    {
        printf ("\n ERROR WHILE CURSOR DECLARATION");
        printf ("\n RETURNED SQLCODE = %d", SQLCODE);
        rc = -1;
        exit(1);
    }

    /* open cursor */
(2) EXEC SQL OPEN cur1;

    /* fetch cursor */
(3) EXEC SQL FETCH cur1
    INTO :hempno, :hfirstname, :hlastname, :hedlevel, :hsalary ;
    printf("\n\nEMPNO    FIRSTNAME    LASTNAME    EMPSALARY");
    printf("\n-----    -            -            -\n");
    while (SQLCODE != 100)
    {
        printf("%d %10s %11s %15f \n", hempno, hfirstname, hlastname,
            hsalary);
        EXEC SQL FETCH cur1
        INTO :hempno, :hfirstname, :hlastname, :hedlevel, :hsalary;
        if(SQLCODE < 0)
        {
            printf ("\n ERROR WHILE FETCHING DATA");
            printf ("\n RETURNED SQLCODE = %d", SQLCODE);
            rc = -1;
            exit(1);
        }
    }
(4) EXEC SQL CLOSE cur1;
    EXEC SQL COMMIT;
    return rc;
}

```

### Listing 5.7 – SELECT employee information

In the above listing:

1. Declares the cursor cur1
2. Opens the cursor cur1
3. Fetches a row one at a time and prints the values.

4. Closes the cursor.

Figure 5.11 provides the output of above code snippet.

```

=====
EMPLOYEE MANAGEMENT SYSTEM
=====
1. Add New Employee
2. Update Employee Information(Salary)
3. Fetch Employee Information
4. Delete Employee Information
5. Exit

Enter option: 3

EMPNO      FIRSTNAME  LASTNAME   EMPSALARY
-----
50001      RAUL       CHONG      6000.000000
50002      JAMES      JI         5786.000000
50003      MIN        YAO        5876.000000
50004      IAN        HAKES      5489.000000
50005      VINEET    MISHRA     5600.000000

```

Figure 5.11 – Output of SELECT operation

#### 5.3.4.4.3 Updating data in a table

Let's take a look at how you can update data in a table by reviewing the function `UpdateEmployeeInfo()` shown in *Listing 5.8*. This function updates information in the `EMPLOYEE` table for employees whose employee id and new salary is not known.

In this example we use dynamic SQL statements. As discussed in *Chapter 1, Introduction to DB2 Application Development*, in a dynamic SQL statement not everything is known about the SQL structure until runtime. The statement uses parameter markers, indicated by a question mark (?), to indicate the unknowns. The `PREPARE` statement 'compiles' the dynamic SQL.

```

int UpdateEmployeeInfo()
{
    int rc = 0, noofemp, loop;
    printf("\n=====");
    printf("\n UPDATE EMPLOYEE INFORMATION");
    printf("\n=====");
(1) strcpy(hdynstmt, "UPDATE employee SET SALARY = ? WHERE empno
    = ?");
(2) EXEC SQL PREPARE stmt FROM :hdynstmt;
    if(SQLCODE < 0)
    {
        printf ("\n ERROR WHILE PREPARING STATEMENT");
        printf ("\n RETURNED SQLCODE = %d", SQLCODE);
        rc = -1;
        exit(1);
    }
}

```

```

    }
    printf("\n NUMBER OF EMPLOYEE TO UPDATE: ");
    scanf("\n%d", &noofemp);
    for(loop = 0; loop != noofemp; loop++)
    {
        printf("\n ENTER EMPLOYEE ID AND NEW SALARY: ");
        scanf("\n %d %lf",&hempno, &hnewsalary);
(3) EXEC SQL EXECUTE stmt USING :hnewsalary, :hempno;
        if(SQLCODE < 0)
        {
            printf ("\n EROROR WHILE EXECUTING STATEMENT");
            printf ("\n RETURNED SQLCODE = %d", SQLCODE);
            rc = -1;
            exit(1);
        }
    }
    printf("\n EMPLOYEE INFORMATION UPDATED SUCESSFULLY ");
(4) EXEC SQL COMMIT;
    if(SQLCODE < 0)
    {
        printf ("\n EROROR WHILE COMITTING DATA");
        printf ("\n RETURNED SQLCODE = %d", SQLCODE);
        rc = -1;
        exit(1);
    }
    return rc;
}

```

#### Listing 5.8 – UPDATE employee information

In the above listing:

1. Assigns the SQL statement into the variable
2. Prepares the statement with a parameter marker.
3. Executes the statement for host variable **hnewsalary** and **hempno** entered by the user.
4. Commits the transaction.

Figure 5.12 provides the output of the above code snippet.

```

=====
EMPLOYEE MANAGEMENT SYSTEM
=====
1. Add New Employee
2. Update Employee Information(Salary)
3. Fetch Employee Information
4. Delete Employee Information
5. Exit

Enter option: 2

=====
UPDATE EMPLOYEE INFORMATION
=====
NUMBER OF EMPLOYEE TO UPDATE: 1

ENTER EMPLOYEE ID AND NEW SALARY: 50004
5897

EMPLOYEE INFORMATION UPDATED SUCESSFULLY

```

Figure 5.12 - Output of SELECT operation

#### 5.3.4.4.4 Deleting data from a table

Let's take a look at the `DeleteEmployeeInfo()` function which deletes employee information from the EMPLOYEE table. *Listing 5.9* shows the delete example.

```

int DeleteEmployeeInfo()
{
    int rc = 0;
    int option;
    char diagooption;
    printf("\n=====");
    printf("\n DELETE EMPLOYEE INFORMATION");
    printf("\n=====");
    printf("\n ENTER 1: (TO DELETE EMPLOYEE INFORMATION)  ");
    printf("\n ENTER 2: (TO DELETE ALL EMPLOYEES INFORMATION)  ");
    scanf("\n%d", &option);
    if(option == 1)
    {
        printf("\n ENTER EMPLOYEE ID:");
        scanf("\n%d", &hempno);
(1) EXEC SQL DELETE employee WHERE empno = :hempno;
        if(SQLCODE < 0)
        {
            printf ("\n EROROR WHILE DELETING INFORMATION");
            printf ("\n RETURNED SQLCODE = %d", SQLCODE);
            rc = -1;
            exit(1);
        }
        printf("\n EMPLOYEE WITH ID %d DELETED SUCESSFULLY ", hempno);
    }
}

```

```

    }
    else
    {
(2) EXEC SQL DELETE employee WHERE empno BETWEEN 50001 AND 50005;
        if(SQLCODE < 0)
        {
            printf ("\n ERROR WHILE DELETING INFORMATION");
            printf ("\n RETURNED SQLCODE = %d", SQLCODE);
            rc = -1;
            exit(1);
        }
        printf("\n ALL EMPLOYEES DELETED SUCESSFULLY ");
(3) EXEC SQL COMMIT;
    }
    return rc;
}

```

#### Listing 5.9 – DELETE employee information

In the above listing:

1. SQL statement to delete information of a particular employee
2. SQL statement to delete information of all the employees
3. Commit the transaction.

Figure 5.13 provides the output of above code snippet.

```

Enter option: 4
=====
DELETE EMPLOYEE INFORMATION
=====
ENTER 1: <TO DELETE EMPLOYEE INFORMATION>
ENTER 2: <TO DELETE ALL EMPLOYEES INFORMATION> 2
ALL EMPLOYEES DELETED SUCESSFULLY

=====
EMPLOYEE MANAGEMENT SYSTEM
=====
1. Add New Employee
2. Update Employee Information(Salary)
3. Fetch Employee Information
4. Delete Employee Information
5. Exit

Enter option: 3

EMPNO      FIRSTNAME  LASTNAME   EMPSALARY
-----

```

Figure 5.13 – Output of DELETE operation

### 5.3.4.5 Commit statements

After the execution of SQL statements, depending on your code logic, you should code a commit or rollback statement. *Listing 5.10* shows the commit statement.

```
EXEC SQL COMMIT;
if (SQLCODE < 0)
{
    printf ("\n COMMIT ERROR");
    printf ("\n SQLCODE = %d", SQLCODE);
    exit(1);
}
```

**Listing 5.10 – COMMITstatement**

### 5.3.4.6 Disconnecting from the database

Disconnecting from a database is the final step in working with a database. *Listing 5.11* shows the statement to close the database connection.

```
/* Disconnect from the sample database */
EXEC SQL CONNECT RESET;
/* Error handling to check whether disconnection is successful */
if (SQLCODE < 0)
{
    printf ("\n Error while disconnecting from database");
    printf ("\n Returned SQLCODE = ", SQLCODE);
    exit (1);
}
else
{
    printf ("\n DISCONNECT FROM SAMPLE DATABASE SUCCESSFULLY \n\n");
}
```

**Listing 5.11 – Disconnect from the database**

You can also use `DbDisconn` utility to disconnect from the database. For this, you need to include the `utilemb.h` header file in the header section. *Listing 5.12* shows the use of `DbDisconn` utility function.

```
/* disconnect from database */
rc = DbDisconn(dbAlias);
if (rc != 0)
{
    return rc;
}
```

**Listing 5.12 – Disconnect from the database using DbDisconn utility**



### 5.3.5 Building embedded SQL C/C++ applications

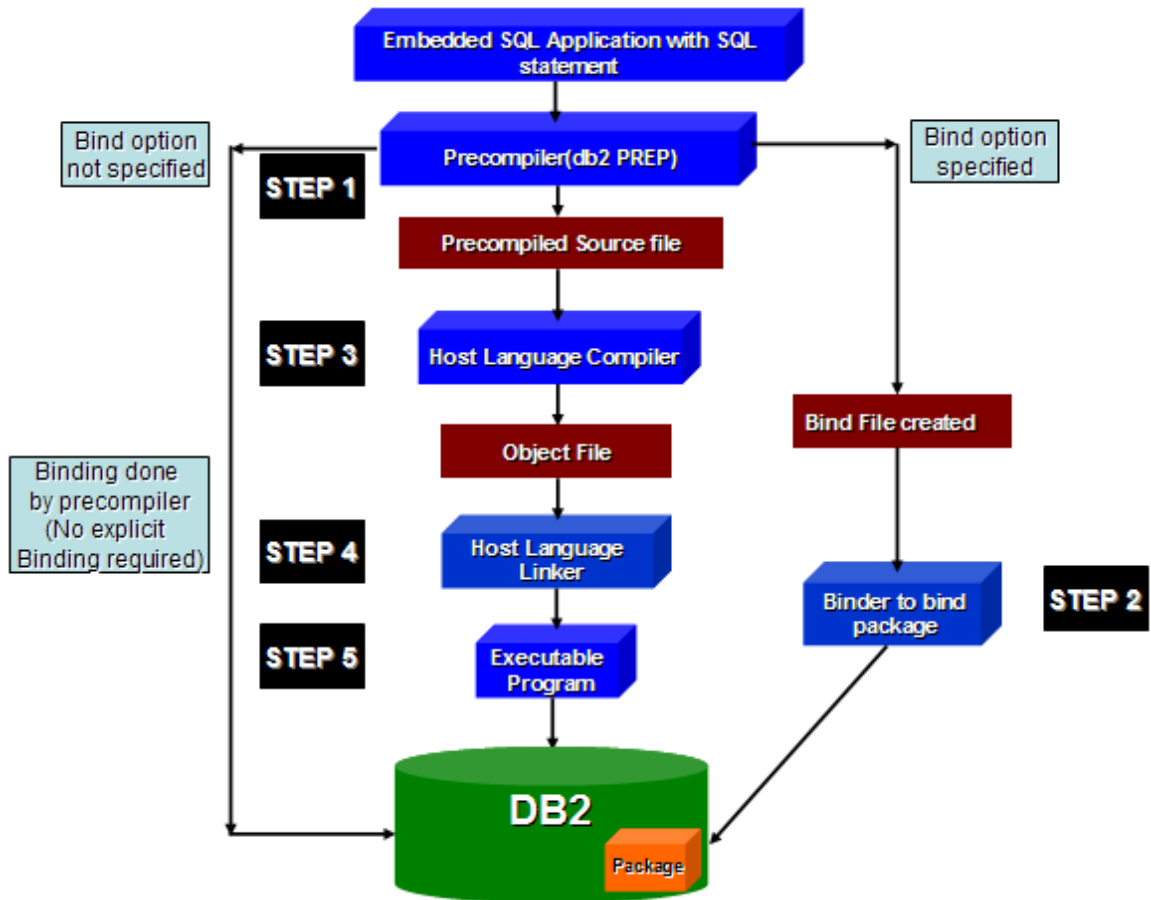
You can build an embedded SQL C/C++ application either manually from the command line or by using DB2 provided scripts. This section discusses both ways.

#### 5.3.5.1 Building embedded SQL C/C++ applications from the command line

Building embedded SQL C/C++ applications from the command line involves the following steps:

1. Precompile the application by issuing the PRECOMPILE command
2. If you created a bind file (by using the BINDFILE option in the PRECOMPILE command in step 1), bind this file to a database to create an application package by issuing the BIND command.
3. Compile the modified application source and the source files that do not contain embedded SQL to create an application object file (a .obj file).
4. Link the application object files with the DB2 and host language libraries to create an executable program using the link command.

These steps are illustrated in *Figure 5.14* below taken from the developerWorks Article *DB2 packages: Concepts, examples, and common problems*. [4]



**Figure 5.14 – Building an embedded C/C++ SQL program**

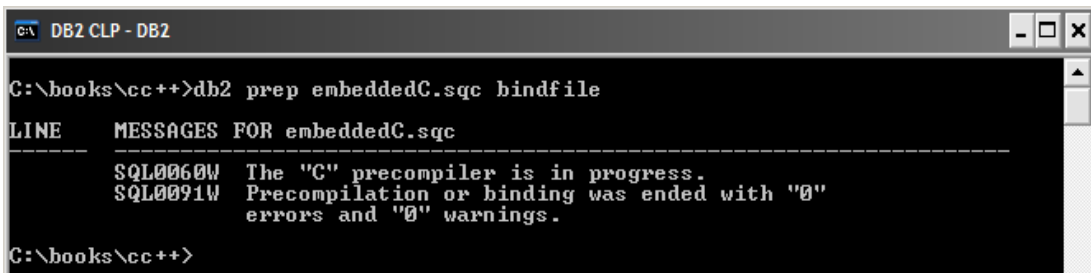
Let's take a look at each of the steps illustrated in *Figure 5.14* in more detail:

#### 5.3.5.1.1 Step 1: Precompile the source file

Once you have created the embedded SQL application's source files, you must precompile each host language file containing SQL statements with the PREP command. The precompiler comments out all SQL statements contained in the source file, generates the DB2 run-time API calls for those statements and creates a BIND file when the BINDFILE option is used as shown below:

```
db2 prep embeddedC.sqc bindfile
```

*Figure 5.15* shows the output of above command



```
c:\ DB2 CLP - DB2
C:\books\cc++>db2 prep embeddedC.sqc bindfile
LINE      MESSAGES FOR embeddedC.sqc
-----
SQL0060W  The "C" precompiler is in progress.
SQL0091W  Precompilation or binding was ended with "0"
          errors and "0" warnings.
C:\books\cc++>
```

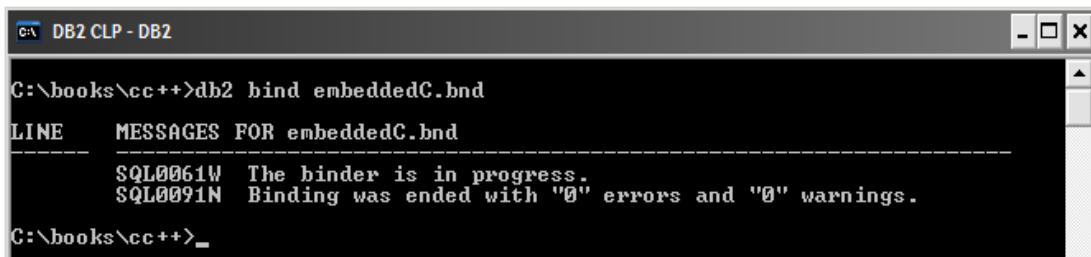
Figure 5.15 – Precompiling a C source file using the PREP command

### 5.3.5.1.2 Step 2: Binding the bind file

Binding is the process of creating a package on the database server out of a bind (.bnd) file that was created by the PREP command. The bind file contains the information required by DB2 such as the collection id, package name, timestamp, host variables, SQL statements, and so on, to create the package on the server. For example:

```
db2 bind embeddedC.bnd
```

Figure 5.16 provides the output of the above command.



```
c:\ DB2 CLP - DB2
C:\books\cc++>db2 bind embeddedC.bnd
LINE      MESSAGES FOR embeddedC.bnd
-----
SQL0061W  The binder is in progress.
SQL0091N  Binding was ended with "0" errors and "0" warnings.
C:\books\cc++>_
```

Figure 5.16 – Binding a package using the BIND command

DB2 also provides a tool called `db2bfd` that can dump the contents of a bind file. This tool can be helpful if you want to get information about the package that it would create, without actually creating the package first.

`db2bfd -b` will dump header information, containing the package name, consistency token, etc. This is shown in Figure 5.17.

```

c:\ DB2 CLP - DB2

embeddedC.bnd: Header Contents

Header Fields:

Field  Value
-----
releaseNum      0x8000
Endian          0x4c
numHvars        7
maxSect         5
numStmt         19
optInternalCnt  4
optCount        9

Name            Value
-----
Isolation Level Cursor Stability
Creator         "ADMINISTRATOR"
App Name        "EMBEDDED"
Timestamp       "oArITZKZ:2009/10/25 19:37:43:40"
Cnulreqd       Yes
Sql Error       No package
Validate        Bind
Date            Default/local
Time            Default/local

```

Figure 5.17 – Output of db2bfd -b

db2bfd -s will dump the SQL statements as shown in Figure 5.18.

```

c:\ DB2 CLP - DB2

C:\books\cc++>db2bfd -s embeddedC.bnd
embeddedC.bnd: SQL Statements = 19

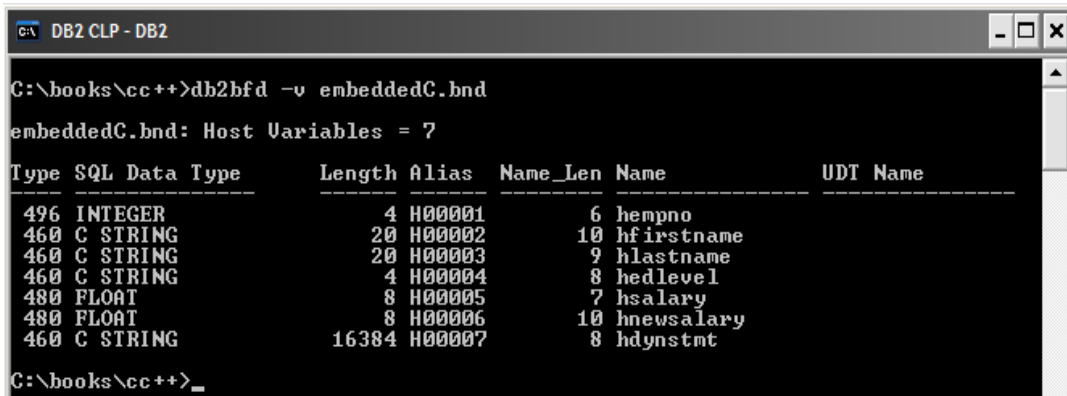
Line  Sec  Typ  Var  Len  SQL statement text
-----
   7    0    5    0   21  BEGIN DECLARE SECTION
  15    0    2    0   19  END DECLARE SECTION
  19    0   10    0   13  INCLUDE SQLCA
  40    0   19    0   17  CONNECT TO SAMPLE
  84    0   19    0   13  CONNECT RESET
 109    1    0    0  253  INSERT INTO employee(empno, firstnm, lastname, edlevel,
      salary) VALUES (50001, 'RAUL', 'CHONG', 21, 60000), (50
      002, 'JAMES', 'JI', 20, 5786), (50003, 'MIN', 'YAO', 20,
      5876), (50004, 'IAN', 'HAKES', 19, 5489), (50005, 'VIN
      EET', 'MISHRA', 19, 5600)
 129    0    8    0    6  COMMIT
 138    2   11    1   27  PREPARE STMT FROM :H00007
 156    2    9    2   39  EXECUTE STMT USING :H00006 , :H00001
 170    0    8    0    6  COMMIT
 188    3   15    0  123  DECLARE CUR1 CURSOR FOR SELECT empno, firstnm, lastname,
      edlevel, salary FROM employee WHERE empno BETWEEN 50001
      AND 50005
 201    3    4    0    9  OPEN CUR1
 203    3    7    5   70  FETCH CUR1 INTO :H00001, :H00002 , :H00003 , :H00004
      :H00005
 210    3    7    5   69  FETCH CUR1 INTO :H00001, :H00002 , :H00003 , :H00004
      :H00005
 220    3    6    0   10  CLOSE CUR1
 221    0    8    0    6  COMMIT
 261    4    0    1   37  DELETE employee WHERE empno = :H00001
 264    5    0    0   51  DELETE employee WHERE empno BETWEEN 50001 AND 50005
 265    0    8    0    6  COMMIT

C:\books\cc++>

```

Figure 5.18 – Output of db2bfd -s

db2bfd -v will dump the host variables as shown in Figure 5.19



```

CA DB2 CLP - DB2
C:\books\cc++>db2bfd -v embeddedC.bnd
embeddedC.bnd: Host Variables = 7
-----
Type  SQL Data Type      Length Alias  Name_Len Name          UDT Name
-----
496  INTEGER                4  H00001      6  hempno
460  C STRING                20  H00002     10  hfirstname
460  C STRING                20  H00003      9  hlastname
460  C STRING                4  H00004      8  hedlevel
480  FLOAT                   8  H00005      7  hsalary
480  FLOAT                   8  H00006     10  hnewsalary
460  C STRING               16384 H00007      8  hdynstmt
C:\books\cc++>_

```

Figure 5.19 – Output of db2bfd -v

### 5.3.5.1.3 Step 3: Compile the modified source file (.c) generated in Step 2

Using your C/C++ compiler, compile the .c file. For example:

```
cl -zi -Od -c -W2 -DWIN32 embeddedC.c
```

Figure 5.20 provides the output of above command



```

CA DB2 CLP - DB2
C:\books\cc++>cl -zi -Od -c -W2 -DWIN32 embeddedC.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.01 for 80x86
Copyright (C) Microsoft Corporation. All rights reserved.
embeddedC.c
C:\books\cc++>

```

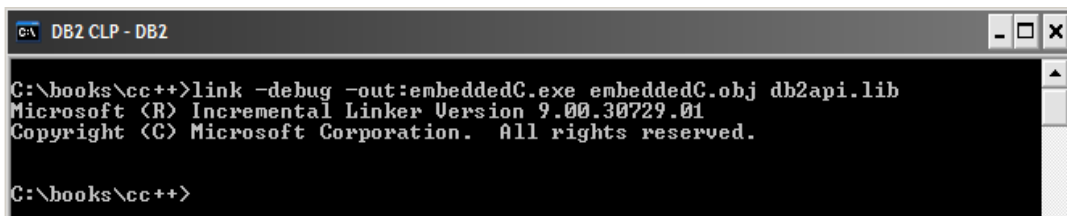
Figure 5.20 – Compile the application

### 5.3.5.1.4 Step 4: Link the file with DB2 and C libraries

Link the .obj files and the DB2 and C libraries as follows:

```
link -debug -out:embeddedC.exe embeddedC.obj db2api.lib
```

Figure 5.21 provides the output of above command.



```

CA DB2 CLP - DB2
C:\books\cc++>link -debug -out:embeddedC.exe embeddedC.obj db2api.lib
Microsoft (R) Incremental Linker Version 9.00.30729.01
Copyright (C) Microsoft Corporation. All rights reserved.
C:\books\cc++>

```

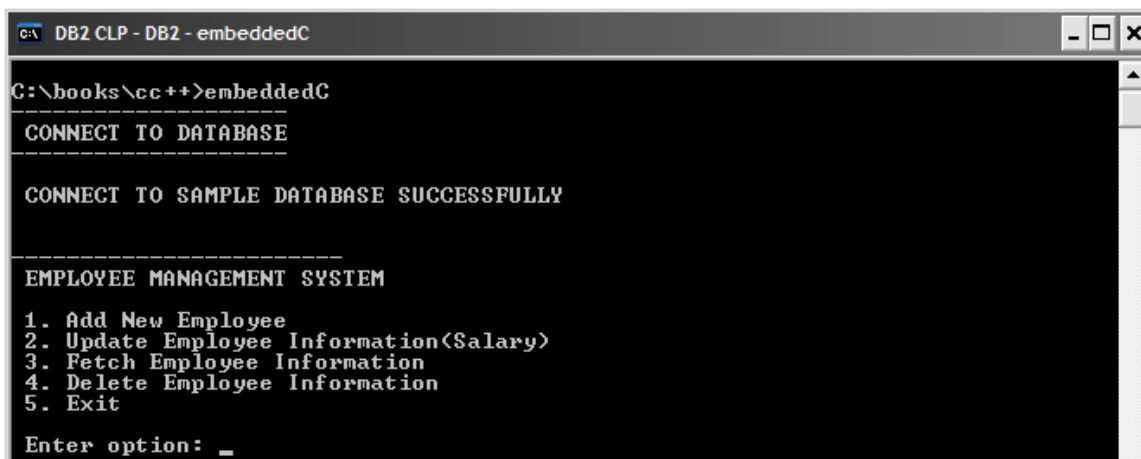
Figure 5.21 – Link application with DB2 libraries

### 5.3.5.1.5 Step 5: Execute the application

The result of step 4 is an executable file, `embeddedC`. You can run the executable file by entering the executable name (On Windows), or changing the permissions in Linux so it's an executable file (`chmod +x`):

```
embeddedC
```

Figure 5.22 provides the output of above command.



```
c:\ DB2 CLP - DB2 - embeddedC
C:\books\cc++>embeddedC
-----
CONNECT TO DATABASE
-----
CONNECT TO SAMPLE DATABASE SUCCESSFULLY
-----
EMPLOYEE MANAGEMENT SYSTEM
1. Add New Employee
2. Update Employee Information(Salary)
3. Fetch Employee Information
4. Delete Employee Information
5. Exit
Enter option: _
```

Figure 5.22 – Output of the executable file `embeddedC`

### 5.3.5.2 Building embedded SQL C/C++ applications using the sample build script

To build an embedded C application, the easiest way is to use `bldapp` script provided by DB2. The `bldapp` script compiles and links the embedded application. The script is located in the `sqllib/samples/c` directory if the application is written in C. If the application is written in C++ the location of the script is `sqllib/samples/cpp`, along with sample utility programs that can be built with these files.

`bldapp` scripts takes up to four parameters, represented inside the script file by the variables `$1`, `$2`, `$3`, and `$4`. The parameter, `$1`, specifies the name of your source file. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: the second parameter, `$2`, specifies the name of the database to which you want to connect; the third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password.

If the program contains embedded SQL, indicated by the `.sql` extension, then the `embprep` script is called to precompile the program, producing a program file with a `.c` extension.

The following example shows how to build and run embedded C applications. To build the above sample program `embeddedC.sql` enter:

```
bldapp embeddedC
```

Figure 5.23 provides the output, and highlights four steps which were also discussed in section 5.4.1.

```

c:\Program Files\IBM\SQLLIB\samples\c>bldapp embeddedC

Database Connection Information
Database server      = DB2/NT 9.7.0
SQL authorization ID = ADMINIST...
Local database alias = SAMPLE

LINE  MESSAGES FOR embeddedC.sqc
-----
SQL0060W The "C" precompiler is in progress.
SQL0091W Precompilation or binding was ended with "0"
         errors and "0" warnings.
LINE  MESSAGES FOR utilenb.sqc
-----
SQL0060W The "C" precompiler is in progress.
SQL0091W Precompilation or binding was ended with "0"
         errors and "0" warnings.
LINE  MESSAGES FOR embeddedC.bnd
-----
SQL0061W The binder is in progress.
SQL0091N Binding was ended with "0" errors and "0" warnings.
DB20000I The SQL command completed successfully.

C:\Program Files\IBM\SQLLIB\samples\c>ren Compile the program.
C:\Program Files\IBM\SQLLIB\samples\c>if exist "embeddedC.cxx" goto cpp_enb
C:\Program Files\IBM\SQLLIB\samples\c>cl -Zi -Od -c -W2 -DWIN32 embeddedC.c utilenb.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.01 for 80x86
Copyright (C) Microsoft Corporation. All rights reserved.
embeddedC.c
utilenb.c
Generating Code...
C:\Program Files\IBM\SQLLIB\samples\c>goto link_embedded
C:\Program Files\IBM\SQLLIB\samples\c>link -debug -out:embeddedC.exe embeddedC.obj utilenb.obj db2api.lib
Microsoft (R) Incremental Linker Version 9.00.30729.01
Copyright (C) Microsoft Corporation. All rights reserved.
C:\Program Files\IBM\SQLLIB\samples\c>goto exit
C:\Program Files\IBM\SQLLIB\samples\c>

```

Figure 5.23 – Output of building the application using bldapp script

The result is an executable file, embeddedC. You can run the executable file by entering the executable name:

```
embeddedC
```

## 5.5 Developing a C/C++ application with ODBC/CLI

In section 1.3.3 you were introduced to ODBC/CLI development. In this chapter we discuss this subject again, but in more detail. DB2 Call Level Interface (DB2 CLI) is IBM's callable SQL interface to the DB2 family of database servers. It is a C and C++ application programming interface for relational database access that uses function calls to pass dynamic SQL statements as function arguments. DB2 CLI is based on the Microsoft Open Database Connectivity (ODBC) specification and the International Standard for SQL/CLI. The DB2 CLI driver is included with all DB2 servers and several clients as discussed in section 1.3.3. It behaves as the ODBC driver for ODBC applications connecting to DB2.

ODBC is an alternative to embedded dynamic SQL, but unlike embedded SQL, it does not require host variables or a precompiler. The main advantage of an ODBC application over

an embedded SQL C/C++ application is that an ODBC application can run against a variety of databases from different vendors without having to compile the code against each of these databases. To run a compiled ODBC/CLI application, you only need to install the ODBC/CLI driver that the vendor provides on the machine where you are running the application, and their client. Applications use procedure calls at run time to connect to databases, issue SQL statements, and retrieve data and status information.

### 5.5.1 Additional environment setup for CLI/ODBC applications

In addition to what was discussed in section 5.2, for an ODBC/CLI application to access a DB2 database you also need to follow these steps:

#### 5.5.1.1 Linux

1. Install an ODBC driver manager.
2. Register the DB2 database as an ODBC data source in the `.odbc.ini` file. For example, to register the SAMPLE database, the `.odbc.ini` file must contain the following line

```
SAMPLE=IBM DB2 ODBC DRIVER
```

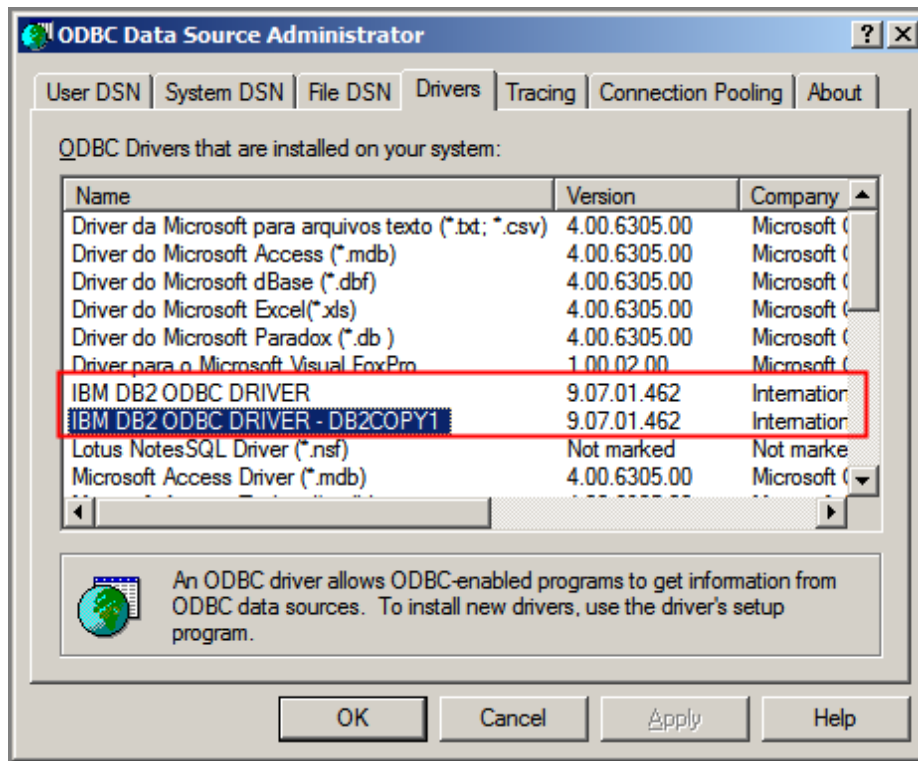
3. Set the `LD_LIBRARY_PATH` environment variable to `libodbc.so`.
4. Set the `ODBCINI` environment variable as follows:

```
ODBCINI=/opt/odbc/system_odbc.ini;export ODBCINI
```

#### 5.5.1.2 Windows

1. Make sure that the Microsoft ODBC Driver Manager and the DB2 CLI/ODBC driver are installed. To verify that both exist on the machine follow the steps below:
  - Double click on the Microsoft ODBC Data Sources icon in the Control Panel, or run the `odbcad32.exe` command from the command line.
  - Click on the Drivers tab and check IBM DB2 ODBC DRIVER - <DB2\_Copy\_Name> is shown in the list. *Figure 5.24* shows the output. If ODBC Driver Manager or the IBM DB2 CLI/ODBC driver is not in the list, you need to download the IBM Data Server driver for ODBC and CLI from <http://www-01.ibm.com/support/docview.wss?rs=4020&uid=swg21385217> and install it by copying the `clidriver` folder to <DB2 install directory>\sqllib





**Figure 5.24 – Add ODBC data source**

- The next step is to configure the ODBC Data Source. Follow below steps to set up the Data Source.
  - In the ODBC Data Source Administrator panel (as shown in Figure 5.24 above), click on the System DSN tab.
  - Click on Add, select IBM DB2 ODBC driver and click finish to create new data source.
  - In the window that pops up, enter any name for the data source name, in this example we will use *sample*, then choose the database SAMPLE and click on *add* button to register this database.
- You can also check if the DB2 database is registered by listing your current data sources with the following command from the DB2 command window or Linux shell:

```
db2 list system odbc data sources
```

Figure 5.25 shows the output of above command. You can see the data source SAMPLE is added in the list.

```

C:\ DB2 CLP - DB2
C:\Documents and Settings\Administrator>db2 list system odbc data sources
System ODBC Data Sources

Data source name          Description
-----
ELCertificateDB          Microsoft Access Driver (*.mdb)
ELRepository              Microsoft Access Driver (*.mdb)
sample                    IBM DB2 ODBC DRIVER - DB2

```

Figure 5.25 – Add ODBC data source

- An alternative to using the ODBC Data Source Administrator is to register the database using the command below:

```
db2 catalog system odbc data source sample
```

### 5.5.2 Handles

An ODBC/CLI handle is a variable that refers to a data object allocated and managed by ODBC/CLI. In simpler terms, a handle is a pointer to a variable which is used for passing references to the variable between parts of the program. There are four types of handles in ODBC/CLI:

- **Environment handle (SQL\_HANDLE\_ENV)**

An environment handle refers to an object that holds information about the global state of the application, attributes or connections. An environment handle must be allocated before a connection handle can be allocated.

- **Connection handle (SQL\_HANDLE\_DBC)**

A connection handle refers to an object that holds information about the connection to a particular database. For each database, a separate connection handle must be allocated. A connection handle must be allocated before a statement handle can be allocated.

- **Statement handle (SQL\_HANDLE\_STMT)**

A statement handle refers to an object that holds information about the execution of a single SQL statement. Before the execution of a SQL statement, a statement handle must be allocated and associated with a connection handle.

- **Descriptor handle (SQL\_HANDLE\_DESC)**

A descriptor handle refers to an object that contains information about the columns in a result set and dynamic parameters in an SQL statement.

Figure 5.26 taken from the IBM redbook *DB2 Express-C: The Developer Handbook for XML, PHP, C/C++, Java, and .NET* [5] illustrates the relationship between the different handles.

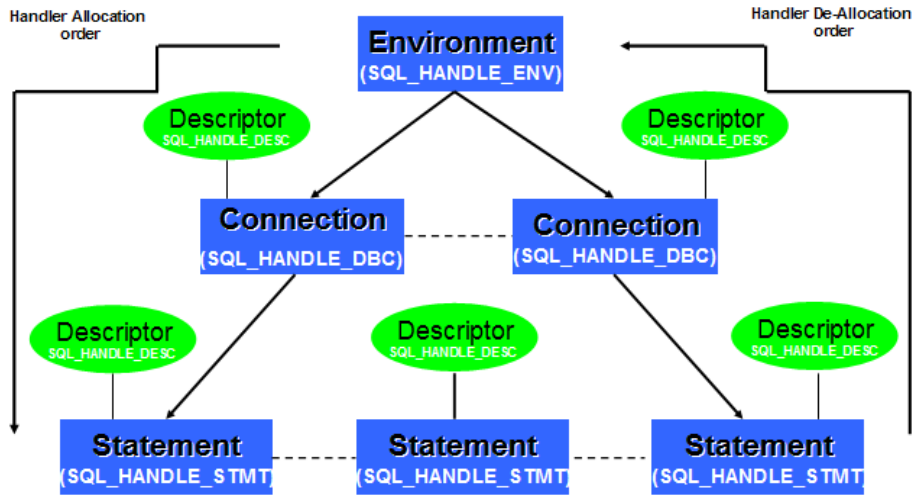


Figure 5.26 – CLI Handles

### 5.5.3 Steps to develop an ODBC/CLI application

Figure 5.27 shows the steps required to develop an ODBC/CLI application.

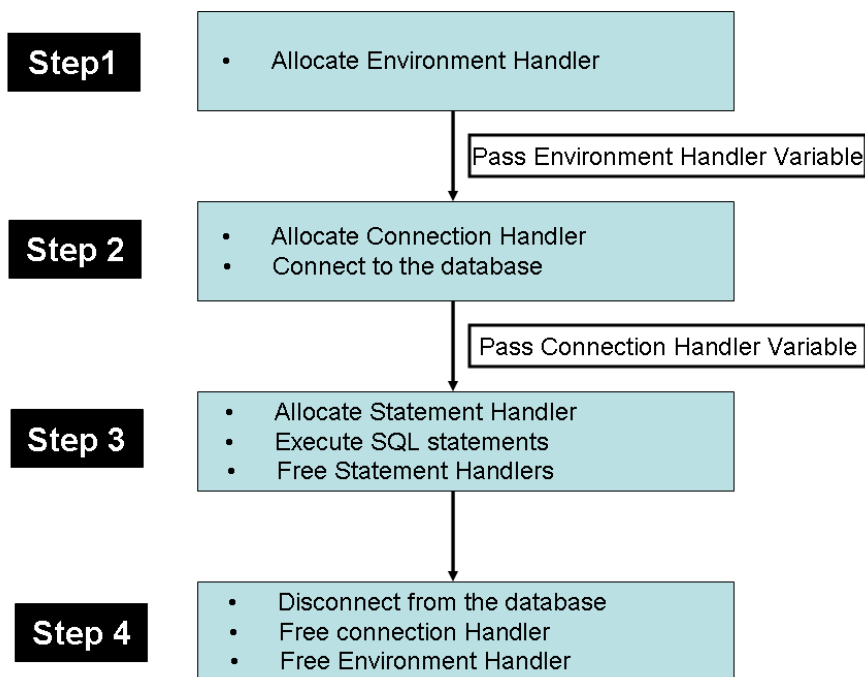


Figure 5.27 – Steps required in an ODBC/CLI application

The steps shown in Figure 5.27 are described in more detail in the following sections.

### 5.5.3.1 Include header files

To start application development with CLI, you need to include `sqlcli1.h` header file which contains CLI constants, function prototypes, data structures required for developing CLI application. *Listing 5.13* lists the header files required for a CLI/ODBC application.

```
/* Include header files */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sqlcli1.h>
```

#### Listing 5.13 – Header files required for CLI application

### 5.5.3.2 Allocate environment handle

Allocation of different handles can be done using `SQLAllocHandle` API. `SQLAllocHandle()` is a generic function that allocates environment, connection, statement, or descriptor handles.

**Note:**

This function replaces the deprecated functions `SQLAllocConnect()`, `SQLAllocEnv()`, and `SQLAllocStmt()`. For a complete list of CLI and ODBC functions, review this link <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.cli.doc/doc/r0000553.html>

*Listing 5.14* shows the syntax of `SQLAllocHandle` API

```
SQLRETURN SQLAllocHandle(
    SQLSMALLINT      HandleType,
    SQLHANDLE        InputHandle,
    SQLHANDLE        *OutputHandlePtr);
```

#### Listing 5.14 – Syntax of `SQLAllocHandle` function

Function `SQLAllocHandle` takes as arguments the type of handle (environment, connection, statement, or descriptor), the input handle and the output handle. To allocate an environment handle the value of `HandleType` must be `SQL_HANDLE_ENV`. If the handle type is `SQL_HANDLE_ENV` the value of `InputHandle` will be `SQL_NULL_HANDLE`. `OutputHandlePtr` will be a pointer to a buffer in which to return the handle to the newly

allocated data structure. After the handle is allocated, `SQLAllocHandle` returns any one of the following return codes, which are self-explanatory:

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_INVALID_HANDLE`
- `SQL_ERROR`

Let's take a look at an example. *Listing 5.15* shows the allocation of an environment handle.

```

/* allocate an environment handle */
(1) rc = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, pHenv);
(2) if (rc != SQL_SUCCESS)
    {
        printf("\n\nERROR while allocating environment handle.\n");
        funcRc = 1;
        exit (1);
    }
    printf("\n\nAllocate environment handle successfully.");

```

#### Listing 5.15 – Allocate Environment handle using `SQLAllocHandle` function

In the above listing:

1. This statement allocates the environment handle using `SQLAllocHandle` API and returns the pointer to `pHenv`.
2. This statement checks whether allocation is successful or not.

#### 5.5.3.3 Connect to the database

After the successful allocation of the environment handle, the next step is to connect to the database using `SQLConnect()`. Before connecting to the database you need to allocate a connection handle. Allocation of a connection handle can be done by the `SQLAllocHandle` API using `SQL_HANDLE_DBC` as the type of handle. The value of `InputHandle` will be the pointer to the environment handle variable. *Listing 5.16* shows the syntax of `SQLConnect`, and *Listing 5.17* provides an example of allocating a connection handle using `SQLAllocHandle` and connecting to a database using `SQLConnect`.

```

SQLRETURN SQLConnect (
    SQLHDBC ConnectionHandle, /* hdbc */
    SQLCHAR *ServerName, /* szDSN */
    SQLSMALLINT ServerNameLength, /* cbDSN */
    SQLCHAR *UserName, /* szUID */
    SQLSMALLINT UserNameLength, /* cbUID */
    SQLCHAR *Authentication, /* szAuthStr
*/

```

```
SQLSMALLINT AuthenticationLength); /* cbAuthStr */
```

### Listing 5.16 – Syntax of SQLConnect function

```
/* allocate a database connection handle */
(1) rc = SQLAllocHandle(SQL_HANDLE_DBC, *pHenv, pHdbc);
    if (rc != SQL_SUCCESS)
    {
        printf("\n\nERROR while allocating connection handle.\n");
        funcRc = 1;
        exit (1);
    }
    printf("\n\nAllocate Connection handle successfully.");

/* connect to the database */
(2) rc = SQLConnect(*pHdbc, (SQLCHAR *)dbAlias, SQL_NTS,(SQLCHAR
    *)user,SQL_NTS,(SQLCHAR *)pswd, SQL_NTS);
    if (rc != SQL_SUCCESS)
    {
        printf("\n\nERROR while connecting to database.\n");
        funcRc = 1;
        exit (1);
    }
    printf("\n\nConnected to %s database successfully\n", dbAlias);
```

### Listing 5.17 – Allocate connection handle and connect to the database

In the above listing:

1. This statement allocates the connection handle using SQLAllocHandle function.
2. This statement connects to the database using SQLConnect. Database name will be passed as an argument.

If you would like to test the above code snippet, copy the **ApplInit** function from `cli_odbc.c` program (accompanying this book) and change the userid, and password in the program. The program includes the handle allocation and connection statements shown in *Listing 5.15* and *Listing 5.17*. *Figure 5.28* provides the output of above code snippet of *Listing 5.15* and *5.17*.



```

DB2 CLP - DB2 - cli_odbc
C:\books\cli>cli_odbc

Allocate environment handle successfully.
Allocate Connection handle successfully.
Connected to sample database successfully

```

**Figure 5.28 – Connection handle allocation and connect to the database**

There are other APIs available that can be used to connect to a database. The `SQLDriverConnect()` API extends the functionality of `SQLConnect()` by adding extra connection parameters and the ability to get connection information from the user. *Table 5.4* lists the available connection related APIs.

| CLI Connection API             | Use of API                                        |
|--------------------------------|---------------------------------------------------|
| <code>SQLConnect</code>        | Establishes the connection to the target database |
| <code>SQLBrowseConnect</code>  | Get required attributes to connect to data source |
| <code>SQLSetConnectAttr</code> | Set the connection attributes                     |
| <code>SQLGetConnectAttr</code> | Get the current attribute setting                 |
| <code>SQLDisconnect</code>     | Disconnect from the data source                   |

**Table 5.4 – Connection related CLI APIs**

Performing the above steps every time on a large application would be very time consuming. To make things easier, you can use utility functions provided by DB2 in the samples directory. You need to include `utilcli.h` header file in the header section of your application. Let's take a look at an example where you can use the utility function `CLIAppInit()` to allocate the environment handle, connection handle, set `AUTOCOMMIT`, and connect to the database. *Listing 5.18* illustrates this.

```

/* initialize the CLI application by calling a helper utility function
defined in utilcli.c */
rc = CLIAppInit(dbAlias, user, pswd, &henv, &hdbc,
(SQLPOINTER)SQL_AUTOCOMMIT_ON);
if (rc != 0)
{
    return rc;
}

```

**Listing 5.18 – Initialize CLI application using utility function**

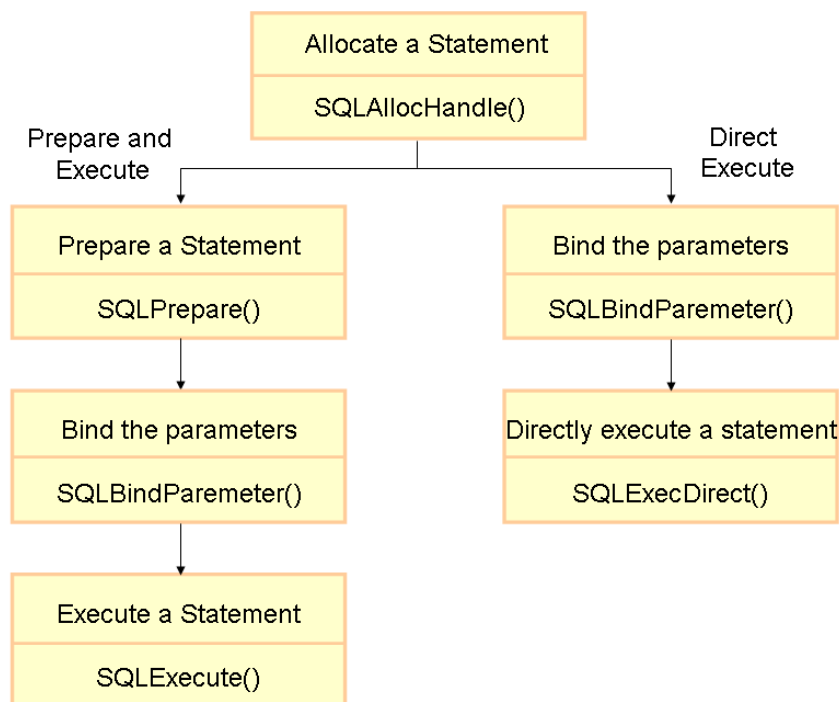
The above code will perform the initialization of a CLI application taking as arguments the database name (alias), userid, password, environment handle variable and connection handle variable as parameters.

### 5.5.3.4 Processing SQL statements

In CLI to issue a SQL statement, your first need to allocate a statement handle. A statement handle is associated with a connection handle and tracks the execution of a SQL statement. Processing SQL statements requires four steps:

1. Allocation of the statement handle
2. Preparing and executing SQL statements
3. Processing the results
4. Freeing the allocated statement handle.

Figure 5.29 illustrates the steps required for the processing of a SQL statement.



**Figure 5.29 – Processing of a SQL statement**

SQL statements are passed as SQLCHAR string variable to DB2 CLI functions. The variable contains a SQL statement with or without parameter markers. Listing 5.19 provides an example.

```

/* SQL INSERT statement to be executed */
SQLCHAR *stmt = (SQLCHAR *)"INSERT INTO employee(empno, firstnme,
lastname, edlevel, salary) VALUES (50006, 'SANJAY', 'KUMAR', 21, 50000),
(50007, 'RAJIT', 'PILLAI', 19, 47860), (50008, 'PRIYANKA', 'JOSHI', 20,
45600)";
  
```



**Listing 5.19 – Storing a SQL statement in a string variable**

In the above example, the *stmt* variable is assigned the INSERT statement. You can also pass a SQL string argument cast to an SQLCHAR \* to the function that will use the SQL directly, without the need of a variable. This is shown in *Listing 5.20* where the function **SQLExecDirect** directly uses the SQL statement.

```
/* SQL INSERT statement to be executed */
SQLExecDirect (hstmt, (SQLCHAR *) " INSERT INTO employee(empno, firstme,
lastname, edlevel, salary) "
"VALUES (50006, 'SANJAY', 'KUMAR', 21, 50000), "
"(50007, 'RAJIT', 'PILLAI', 19, 47860), "
"(50008, 'PRIYANKA', 'JOSHI', 20, 45600)", SQL_NTS);
```

**Listing 5.20 – Directly using a SQL statement in a function**

Table 5.5 provides the list of supported CLI APIs for processing SQL statements.

| CLI APIs         | Use Of API                                         |
|------------------|----------------------------------------------------|
| SQLPrepare       | Prepare a statement                                |
| SQLBindParameter | Bind a parameter marker to a buffer                |
| SQLSetParam      | Bind a parameter marker to a buffer                |
| SQLDescribeParam | Return description of a parameter marker           |
| SQLExecute       | Execute a statement                                |
| SQLExecDirect    | Execute a statement directly                       |
| SQLNumParams     | Get number of parameters in a SQL statement        |
| SQLNumResultCols | Get number of result columns                       |
| SQLBindCol       | Bind a column to an application variable           |
| SQLFetch         | Fetch next row                                     |
| SQLGetDiagField  | Get a field of diagnostic data                     |
| SQLEndTran       | End transactions of a connection or an Environment |

**Table 5.5 – List of supported CLI APIs for processing SQL statements**

**Note:**

This list is not a complete list. For more details and the latest information about the supported CLI APIs, refer to the DB2 9.7 information center at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.cli.doc/doc/r0000553.html>

The examples below demonstrate how to use CLI APIs listed in *Table 5.5* for INSERT, SELECT and DELETE operations.

**5.5.3.4.1 Inserting data**

Function `AddEmployeeInfo()` adds new employee information in the EMPLOYEE table. To execute the INSERT SQL statement directly, the function uses `SQLExecDirect` CLI function as shown in *Listing 5.21*.

```

/* Perform INSERT operation */
int AddEmployeeInfo(SQLHANDLE hdbc)
{
    int funcRc = 0;
    SQLRETURN rc = SQL_SUCCESS;
    SQLHANDLE hstmt; /* statement handle */

    /* SQL INSERT statement to be executed */
(1)  SQLCHAR *stmt = (SQLCHAR *)"INSERT INTO employee(empno, firstnme,
    lastname, edlevel, salary) "
    "VALUES (50006, 'SANJAY', 'KUMAR', 21, 50000), "
    "(50007, 'RAJIT', 'PILLAI', 19, 47860),"
    "(50008, 'PRIYANKA', 'JOSHI', 20, 45600)";

    /* allocate a statement handle */
(2)  rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    if(rc != SQL_SUCCESS)
    {
        printf("\n Error while allocating statement handle");
        funcRc = 1;
        exit (1);
    }

    /* execute the statement */
(3)  rc = SQLExecDirect(hstmt, stmt, SQL_NTS);
    if(rc != SQL_SUCCESS)
    {
        printf("\n Error while statement execution");
        funcRc = 1;
        exit (1);
    }
}

```

```
    }

    /* Commit the transaction */
(4) rc = SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_COMMIT);
    if(rc != SQL_SUCCESS)
    {
        printf("\n Error while committing the transaction");
        funcRc = 1;
        exit (1);
    }

    /* free the statement handle */
(5) rc = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    if(rc != SQL_SUCCESS)
    {
        printf("\n Error while freeing handle");
        funcRc = 1;
        exit (1);
    }
    printf("\n Employee added successfully ");
    return funcRc;
}
```

**Listing 5.21 – Insert data into the table**

In the above listing:

1. *stmt* variable contains the SQL INSERT statement.
2. Allocates the statement handle using **SQLAllocHandle()** API. Connection handle and statement handle info are passed as arguments.
3. Executes the statement directly using **SQLExecDirect()** API. Statement handle variable and *stmt* variable are passed as arguments.
4. Commit the transaction using **SQLEndTran()** API. Connection handle variable and **SQL\_COMMIT** are passed as arguments.
5. Frees allocated statement handle using **SQLFreehandle()** API. Statement handle variable passed as an argument.

*Figure 5.30* provides the output of above code snippet.

```

C:\ DB2 CLP - DB2 - cli_odbc
=====
EMPLOYEE MANAGEMENT SYSTEM
=====
1. Add New Employee
2. Fetch Employee Information
3. Delete Employee Information
4. Exit

Enter option: 1

Employee added successfully

```

Figure 5.30- Output of INSERT operation

#### 5.5.3.4.2 Retrieving data

Retrieving query results with the SELECT statement involves binding application variables to columns of a result set and then fetching the rows of data into the application variables. To retrieve rows of the result set you need to

1. Bind application variable to each column of the result set. Binding can be done by using the `SQLBindCol()` function.
2. Repeatedly fetch the row of data from the result set by calling `SQLFetch()` until `SQL_NO_DATA_FOUND` is returned.

*Listing 5.22* illustrates the function `FetchEmployeeInfo()` that shows how to fetch employee information. We will use parameter markers represented with a question mark (?). The application must bind each application variable to a parameter marker in the SQL statement before it can execute the statement. Binding is done by calling the `SQLBindParameter()` function.

```

/* perform Select operation */
int FetchEmployeeInfo(SQLHANDLE hdbc)
{
    int funcRc = 0;
    SQLRETURN rc = SQL_SUCCESS;
    SQLHANDLE hstmt; /* statement handle */

    (1) /* SQL SELECT statement to be executed */
        SQLCHAR *stmt = (SQLCHAR *) "SELECT firstnme, lastname, edlevel FROM
        employee WHERE empno >= ? ";

        sqlint32 empno = 0;

    (2) struct
        {
            SQLINTEGER ind;
            SQLCHAR value[20];

```

```
    } firstname;          /* variable to be bound to the firstnme column */

    struct
    {
        SQLINTEGER ind;
        SQLCHAR value[15];
    } lastname;          /* variable to be bound to the lastname column */

    struct
    {
        SQLINTEGER ind;
        SQLSMALLINT value;
    } edlevel;          /* variable to be bound to the edlevel column */

    /* set AUTOCOMMIT OFF */
(3) rc = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
        (SQLPOINTER)SQL_AUTOCOMMIT_OFF, SQL_NTS);
    if(rc != SQL_SUCCESS)
    {
        printf("\n Error while handle allocation");
        funcRc = 1;
        exit (1);
    }

    /* allocate a statement handle */
(4) rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    if(rc != SQL_SUCCESS)
    {
        printf("\n Error while allocating statement handle ");
        funcRc = 1;
        exit (1);
    }

    /* prepare the statement */
(5) rc = SQLPrepare(hstmt, stmt, SQL_NTS);
    if(rc != SQL_SUCCESS)
    {
        printf("\n Error while preparing statement");
        funcRc = 1;
        exit (1);
    }

    /* bind empno to the statement */
(6) rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
```

```
        SQL_INTEGER, 0, 0, &empno, 0, NULL);
if(rc != SQL_SUCCESS)
{
    printf("\n Error while binding paremeters");
    funcRc = 1;
    exit (1);
}

/* execute the statement for empno = 50006 */
empno = 50006;

/* execute the statement */
(7) rc = SQLExecute(hstmt);
if(rc != SQL_SUCCESS)
{
    printf("\n Error while statement execution");
    funcRc = 1;
    exit (1);
}

/* bind column 1 to variable */
(8) rc = SQLBindCol(hstmt, 1, SQL_C_CHAR, firstname.value, 20,
    &firstname.ind);
if(rc != SQL_SUCCESS)
{
    printf("\n Error while binding column");
    funcRc = 1;
    exit (1);
}

/* bind column 2 to variable */
(9) rc = SQLBindCol(hstmt, 2, SQL_C_CHAR, lastname.value, 15,
    &lastname.ind);
if(rc != SQL_SUCCESS)
{
    printf("\n Error while binding column");
    funcRc = 1;
    exit (1);
}

/* bind column 3 to variable */
(10) rc = SQLBindCol(hstmt, 3, SQL_C_SHORT, &edlevel.value, 0,
    &edlevel.ind);
if(rc != SQL_SUCCESS)
```

```

    {
        printf("\n Error while binding column");
        funcRc = 1;
        exit (1);
    }

    printf("\n\n  FIRSTNAME          LASTNAME      EDLEVEL   \n");
    printf("  -----          -----      -----\n");

    /* fetch each row and display */
(11) rc = SQLFetch(hstmt);
    if (rc == SQL_NO_DATA_FOUND)
    {
        printf("\n Data not found.\n");
    }
    while (rc != SQL_NO_DATA_FOUND)
    {
        printf("      %8s %14s %8d \n", firstname.value, lastname.value,
            edlevel.value);
        /* fetch next row */
        rc = SQLFetch(hstmt);
    }

    /* Commit the transaction */
(12) rc = SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_COMMIT);
    if(rc != SQL_SUCCESS)
    {
        printf("\n Error while committing the transaction");
        funcRc = 1;
        exit (1);
    }

    /* free the statement handle */
(13) rc = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    if(rc != SQL_SUCCESS)
    {
        printf("\n Error while freeing handle");
        funcRc = 1;
        exit (1);
    }
    return funcRc;
}

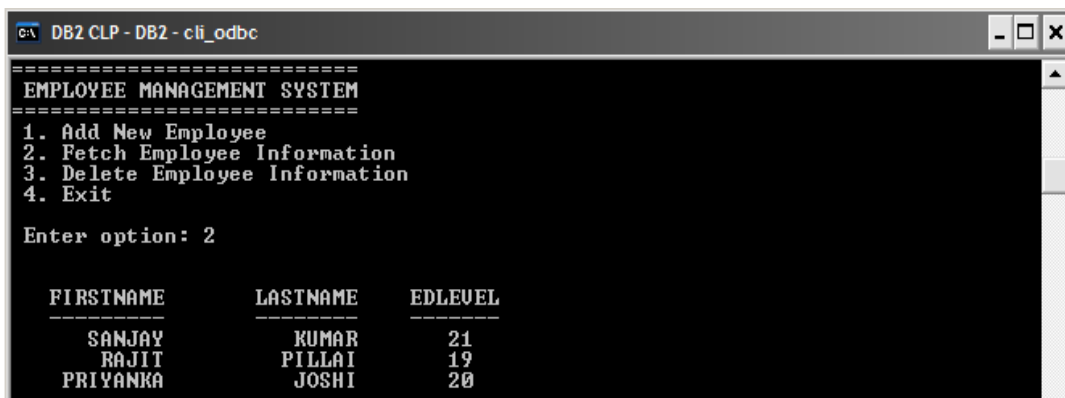
```

**Listing 5.22 – Fetch data from the table**

In the above listing:

1. `stmt` variable contains the SELECT statement
2. Variable to be bound to the firstme, lastname and edlevel column
3. Sets auto commit off using `SQLSetConnectAttr` function.
4. Allocate statement handle using `SQLAllocHandle` function
5. Prepare the SELECT statement using `SQLPrepare` as the statement has parameter markers.
6. Bind `empno` variable to the statement using `SQLBindParameter` function.
7. Execute the statements using `SQLExecute` function for empno 50006.
8. Bind column 1 to variable firstname using `SQLBindCol` function.
9. Bind column 2 to variable lastname using `SQLBindCol` function.
10. Bind column 3 to variable edlevel using `SQLBindCol` function.
11. Fetch each row using `SQLFetch` function and display the results.
12. Commit the transaction using `SQLCommit` function.
13. Free the allocated statement handle using `SQLFreeHandle` function.

Figure 5.31 shows the output of above code snippet



```

=====
EMPLOYEE MANAGEMENT SYSTEM
=====
1. Add New Employee
2. Fetch Employee Information
3. Delete Employee Information
4. Exit

Enter option: 2

FIRSTNAME      LASTNAME      EDLEVEL
-----
SANJAY         KUMAR         21
RAJIT          PILLAI        19
PRIYANKA       JOSHI         20

```

Figure 5.31 – Fetch employee information from the table

#### 5.5.3.4.3 Deleting data

Listing 5.23 shows function `DeleteEmployeeInfo()`. It invokes `SQLExecDirect` to delete employee information.

```

/* Perform Delete operation */
int DeleteEmployeeInfo(SQLHANDLE hdbc)
{
    int funcRc = 0;
    SQLRETURN rc = SQL_SUCCESS;

```



```
SQLHANDLE hstmt; /* statement handle */

/* SQL DELETE statement to be executed */
(1) SQLCHAR *stmt = (SQLCHAR *)"DELETE FROM employee WHERE empno >=
      50006";

/* set AUTOCOMMIT OFF */
(2) rc = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
      (SQLPOINTER)SQL_AUTOCOMMIT_OFF, SQL_NTS);
if(rc != SQL_SUCCESS)
{
    printf("\n Error while setting Auto Commit OFF");
    funcRc = 1;
    exit (1);
}

/* allocate a statement handle */
(3) rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
if(rc != SQL_SUCCESS)
{
    printf("\n Error while allocating statement handle");
    funcRc = 1;
    exit (1);
}

/* directly execute the statement */
(4) rc = SQLExecDirect(hstmt, stmt, SQL_NTS);
if(rc != SQL_SUCCESS)
{
    printf("\n Error while statement execution");
    funcRc = 1;
    exit (1);
}

/* Commit the transaction */
5) rc = SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_COMMIT);
if(rc != SQL_SUCCESS)
{
    printf("\n Error while committing transaction");
    funcRc = 1;
    exit (1);
}

/* free the statement handle */
```

```

6) rc = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
   if(rc != SQL_SUCCESS)
   {
       printf("\n Error while freeing handle");
       funcRc = 1;
       exit (1);
   }
   printf("\n Employee Deleted successfully");
   return funcRc;
}

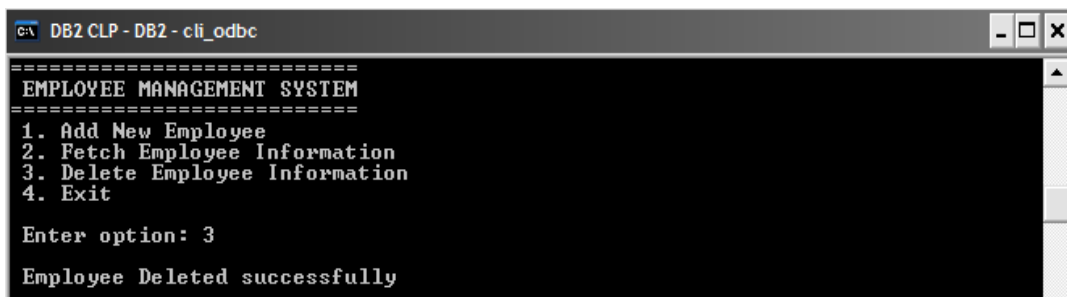
```

### Listing 5.23 – Deleting employee information

In the above listing:

1. *stmt* variable contains the SQL INSERT statement.
2. Sets Auto commit off using `SQLSetConnectAttr` function.
3. Allocates statement handle using `SQLAllocHandle()` function. The connection handle and statement handle are passed as arguments.
4. Executes delete SQL statement directly using `SQLExecDirect()` API. The Statement handle variable and *stmt* variable are passed as arguments.
5. Commit the transaction using `SQLEndTran()` API. The Connection handle and `SQL_COMMIT` are passed as arguments.
6. Frees allocated statement handle using `SQLFreehandle()` API. The Statement handle is passed as an argument.

Figure 5.32 shows the output of the above code snippet.



```

C:\> DB2 CLP - DB2 - cli_odbc
=====
EMPLOYEE MANAGEMENT SYSTEM
=====
1. Add New Employee
2. Fetch Employee Information
3. Delete Employee Information
4. Exit

Enter option: 3

Employee Deleted successfully

```

Figure 5.32 – Delete employee information from the table.

#### 5.5.3.5 Disconnecting from the database

After processing the SQL statements, the next step is to call `SQLDisconnect()` to disconnect from the database. *Listing 5.24* shows how to do this.

```

/* disconnect from the database */
(1) rc = SQLDisconnect(*pHdbc);
(2) if (rc != SQL_SUCCESS)
    {
        printf("\n\nERROR while disconnecting from database.\n");
        funcRc = 1;
        exit (1);
    }
    printf("\n\nDisconnect from %s database successfully.", dbAlias);

```

#### Listing 5.24 – Disconnecting from the database

In the above listing:

1. Disconnect from the database using `SQLDisconnect` API. Pass connection handle variable as an argument.
2. Check whether the disconnection was successful.

#### 5.5.3.6 Freeing the handles

The last step in a CLI application is to free all the allocated handles. To free the allocated handles, DB2 CLI provides the `SQLFreeHandle` API which takes the handle type and the handle variable as arguments. Before calling `SQLFreeHandle()` an application must call `SQLDisconnect()`. Calling `SQLFreeHandle()` before the `SQLDisconnect()` returns `SQL_ERROR` and the connection remains valid. *Listing 5.25* shows how to free the handles.

```

/* free connection handle */
(1) rc = SQLFreeHandle(SQL_HANDLE_DBC, *pHdbc);
    if (rc != SQL_SUCCESS)
    {
        printf("\n\nERROR while freeing connection handle.\n");
        funcRc = 1;
        exit (1);
    }
    printf("\n\nFree connection handle successfully." ) ;

/* free environment handle */
(2) rc = SQLFreeHandle(SQL_HANDLE_ENV, *pHenv);
    if (rc != SQL_SUCCESS)
    {
        printf("\n\nERROR while freeing environment handle.\n");
        funcRc = 1;
        exit (1);
    }
    printf("\n\nFree environment handle successfully.\n\n" ) ;

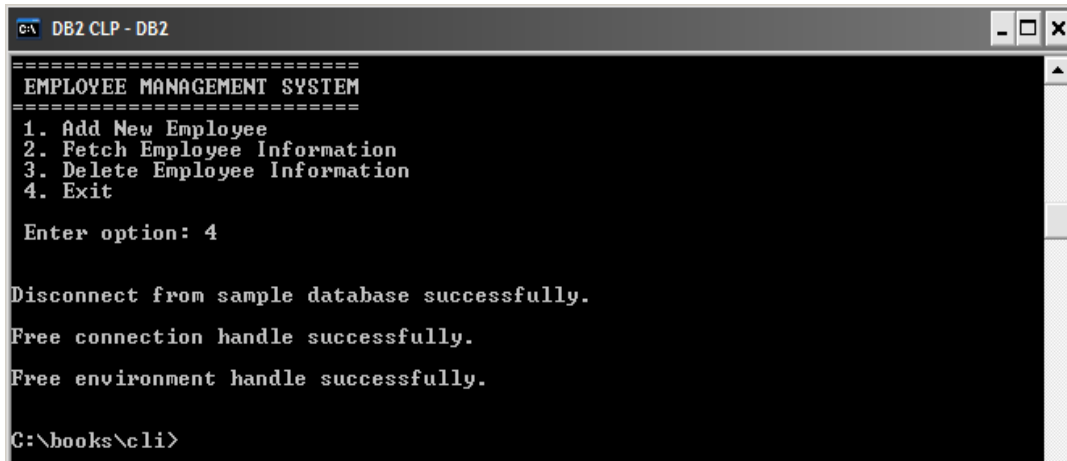
```

**Listing 5.25 – Free allocated environment and connection handles.**

In the above listing:

1. Frees allocated connection handle using `SQLFreehandle`.
2. Frees allocated environment handle using `SQLFreehandle`.

Figure 5.33 shows the output of the above code snippets in Listing 5.24 and 5.25.



```

=====
EMPLOYEE MANAGEMENT SYSTEM
=====
1. Add New Employee
2. Fetch Employee Information
3. Delete Employee Information
4. Exit

Enter option: 4

Disconnect from sample database successfully.
Free connection handle successfully.
Free environment handle successfully.

C:\books\cli>

```

**Figure 5.33 - Freeing allocated handles**

**Note:**

All the code snippets shown in this section are extracted from the program `cli_odbc.c` which is included in the `Exercise_Files_DB2_Application_Development.zip` file with this book. Before building the application ensure to change the `userid` and `password` at lines # 52 and 53 in the program.

**5.5.4 Building ODBC/CLI applications**

You can build ODBC/CLI application either manually from the command line or by using the DB2 provided scripts.

**5.5.4.1 Building an ODBC or CLI application from the command line**

Building ODBC application from the command line involves the following steps:

- 1) Compile the application on windows using below command

```
cl -zi -od -c -w2 -dwin32 cli_odbc.c
```

Figure 5.34 shows the output of above command

```

C:\books\cli>cl -Zi -Od -c -W2 -DWIN32 cli_odbc.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.01 for 80x86
Copyright (C) Microsoft Corporation. All rights reserved.

cli_odbc.c
C:\books\cli>_

```

**Figure 5.34 – Compile application**

2)

Link the application with the CLI library (db2cli.lib) for a CLI application or the odbc library (odbc32.lib) for an ODBC application. For example, the command below would create the CLI application `cli_odbc`:

```
link -debug -out:cli_odbc.exe cli_odbc.obj db2cli.lib db2api.lib
```

Figure 5.35 shows the output of the above command

```

C:\books\cli>link -debug -out:cli_odbc.exe cli_odbc.obj db2cli.lib db2api.lib
Microsoft (R) Incremental Linker Version 9.00.30729.01
Copyright (C) Microsoft Corporation. All rights reserved.

C:\books\cli>_

```

**Figure 5.35 – Linking the application with the CLI library**

On the other hand this command would create an ODBC application:

```
link -debug -out:cli_odbc.exe cli_odbc.obj odbc32.lib
```

Figure 5.36 shows the output of the above command:

```

C:\books\cli>link -debug -out:cli_odbc.exe cli_odbc.obj odbc32.lib
Microsoft (R) Incremental Linker Version 9.00.30729.01
Copyright (C) Microsoft Corporation. All rights reserved.

C:\books\cli>

```

**Figure 5.36 – Link the application with the ODBC library**

You can run the executable file by entering the executable name `cli_odbc`.

#### 5.5.4.2 Building an ODBC or CLI application using the sample build script

To build an application using the sample script `bldapp` that comes with DB2, follow the same recommendations provided earlier in *section 5.4.2*. By default `bldapp` script uses `db2cli.lib` library to build CLI applications. If you want to build ODBC applications using `bldapp` replace `db2cli.lib` with `odbc32.lib` in the script.

To build the sample program `cli_odbc` described earlier, enter:

```
bldapp cli_odbc
```

The result is an executable file, `cli_odbc`. You can run the executable file by entering the executable name:

```
cli_odbc
```

## 5.6 Working with XML in C/C++ applications with DB2

As discussed in *Chapter 2, DB2 pureXML*, DB2 provides the XML data type to store XML data natively. To exchange XML data between the database server and an embedded SQL C/C++ application, you need to declare host variables in your application source code. Columns with the XML data type are described as an `SQL_TYP_XML` column `SQLTYPE`. XML columns can be accessed directly using SQL, XQuery or SQL/XML.

Sample program `xmlinsert.sql`, `xmlread.sql` demonstrates different ways to insert and read the data from an XML column. You can find these sample programs under `<DB2 install path>/samples/xml/c` directory.

## 5.7 Exercises

1. Write a program to read database log files asynchronously with a database connection

Solution: To review the solution check the `dblogconn.sql` sample under `<DB2 install path>/samples/c` directory

2. Write a program to read and write LOB data

Solution: To review the solution check the `dtlob.sql` sample under `<DB2 install path>/samples/c` directory

3. Write a program to use a trigger on a table.

Solution: To review the solution check the `tbtrig.sql` sample under `<DB2 install path>/samples/c` directory

4. Write a program to insert data using the CLI LOAD utility

Solution: To review the solution check the `tbload.c` sample under `<DB2 install path>/samples/cli` directory

## 5.8 Summary

In this chapter you learned the basics of developing C/C++ applications with DB2. The chapter showed you how to set up the environment for building C/C++ applications and how to connect to a DB2 database and issue different kinds of SQL statements using an embedded SQL application and also a CLI/ODBC application. CLI/ODBC applications use procedure calls at run time to connect to databases, issue SQL statements, and retrieve data and status information.

### 5.9 Review questions

1. What is the difference between an embedded SQL C/C++ application and a CLI/ODBC application?
2. What is a cursor?
3. What is a parameter marker?
4. What are handles?
5. What are different return codes returned by CLI APIs?
6. What is the file extension of an embedded SQL C++ program on Windows?
  - A. .sqc
  - B. .c
  - C. .sqx
  - D. .sqC
  - E. None of the above
7. What is the command that can dump the SQL statements from a bind file?
  - A. db2bfd -b
  - B. db2bfd -s
  - C. db2bfd -h
  - D. db2bfd -v
  - E. None of the above
8. What is the command to list current data sources?
  - A. db2 list odbc data sources
  - B. db2 system odbc data sources
  - C. db2 list system odbc data sources
  - D. db2 system odbc data source
  - E. None of the above
9. Which of the following API is used for allocating all the handles?
  - A. SQLAllocHandle ()
  - B. SQLAllocConnect()
  - C. SQLAllocEnv()
  - D. SQLAllocStmt()
  - E. None of the above

10. Which of the following shows the correct flow of handle allocations in CLI application?
- A. Allocate environment handle -> allocate connection handle -> allocate statement handle -> free statement handle-> free connection handle -> free environment handle
  - B. Allocate connection handle -> allocate environment handle -> allocate statement handle -> free statement handle-> free environment handle -> free connection handle
  - C. Allocate connection handle -> allocate environment handle -> free environment handle -> free connection handle
  - D. Allocate environment handle -> allocate statement handle -> free statement handle -> free environment handle
  - E. None of the above



# 6

## Chapter 6 – Application Development with .NET

Microsoft introduced the .NET Framework as a platform for building and executing applications on the Windows platform. It includes a large library that provides many features for Web development, database connectivity, user interface, and so on. With .NET you can write code in over forty different programming languages with C# and Visual Basic being the most popular ones. Regardless of the programming language, .NET applications compile into a type of bytecode that Microsoft calls Intermediate Language (IL), and executes in the Common Language Runtime (CLR). CLR is the heart of the .NET Framework and provides a runtime environment for .NET applications.

In this chapter you will learn about:

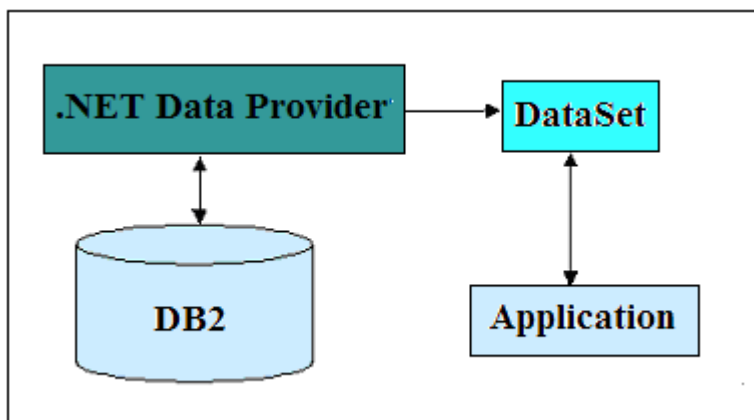
- Setting up the .NET environment to work with DB2
- Understanding DataSet and providers for ADO.NET
- Working with the IBM Database Add-ins for Visual Studio
- Developing .NET with DB2 applications

### 6.1 .NET with DB2 applications: The big picture

In .NET, support to access databases is provided through ActiveX Data Objects (ADO) for .NET. ADO.NET supports both, connected and disconnected database access. .NET applications accessing a database need a .NET data provider which is normally supplied by the database vendor.

For disconnected data access, instances of the DataSet class act as a database cache that resides in your application's memory.

*Figure 6.1* provides an overview of .NET and DB2 applications.



**Figure 6.1 - .NET with DB2 applications**

The figure depicts a .NET application accessing a DB2 database through a .NET data provider supplied with DB2. For disconnected access, it needs to go through a DataSet object.

## 6.2 The ADO.NET data architecture

Data Access in ADO.NET relies on two components: DataSet and Data Provider.

- DataSet

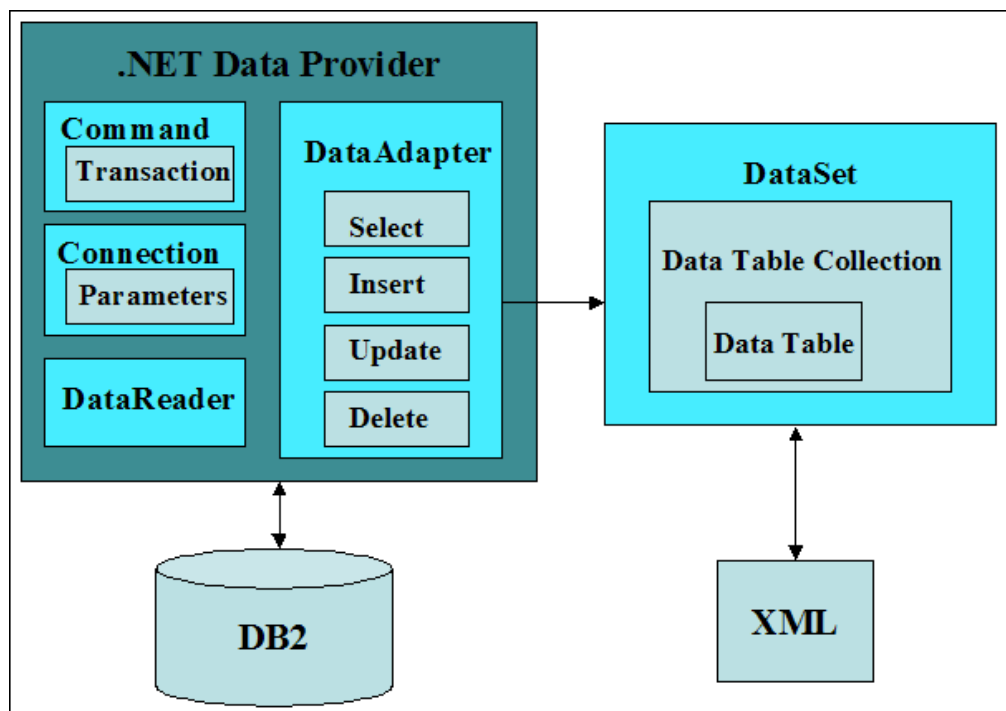
The dataset is a disconnected, in-memory representation of data. It can be considered as a local copy of the relevant portions of the database. The DataSet is persisted in memory and the data in it can be manipulated and updated independently of the database. When the use of a DataSet is finished, changes can be made back to the central database for updating. The data in DataSet can be loaded from any valid data source like DB2.

- Data Provider

The Data Provider is responsible for providing and maintaining the connection to the database. A Data Provider is a set of related components that work together to provide data in an efficient and performance driven manner. Each Data Provider consists of the following component classes:

- The **Connection** object which provides a connection to the database
- The **Command** object which is used to execute a command
- The **DataReader** object which provides a forward-only, read only, connected recordset
- The **DataAdapter** object which populates a disconnected DataSet with data and performs update

Figure 6.2 below shows the relationship between the different ADO.NET objects. These objects will be explained in more detail in the following sections.



**Figure 6.2 - ADO.NET core objects and their relationship**

Data access with ADO.NET can be summarized as follows:

A **Connection** object establishes a connection to the database. A **Command** object executes a query to the database. If the query returns more than a single value, the command object returns a **DataReader**, which is like a cursor. Alternatively, the **DataAdapter** can be used to populate a **DataSet** object. The database can be updated using the **Command** object or the **DataAdapter**.

### 6.2.1 Data providers for ADO.NET

In the ADO.NET architecture, applications -- also known as Data Consumers -- connect to a database -- also referred to as Resource -- using a data provider. The data provider encapsulates data and provides a means to interact with the database including connection, execution of SQL statements, and retrieval of results.

There are 3 types of .NET data providers for DB2 applications to access a DB2 database:

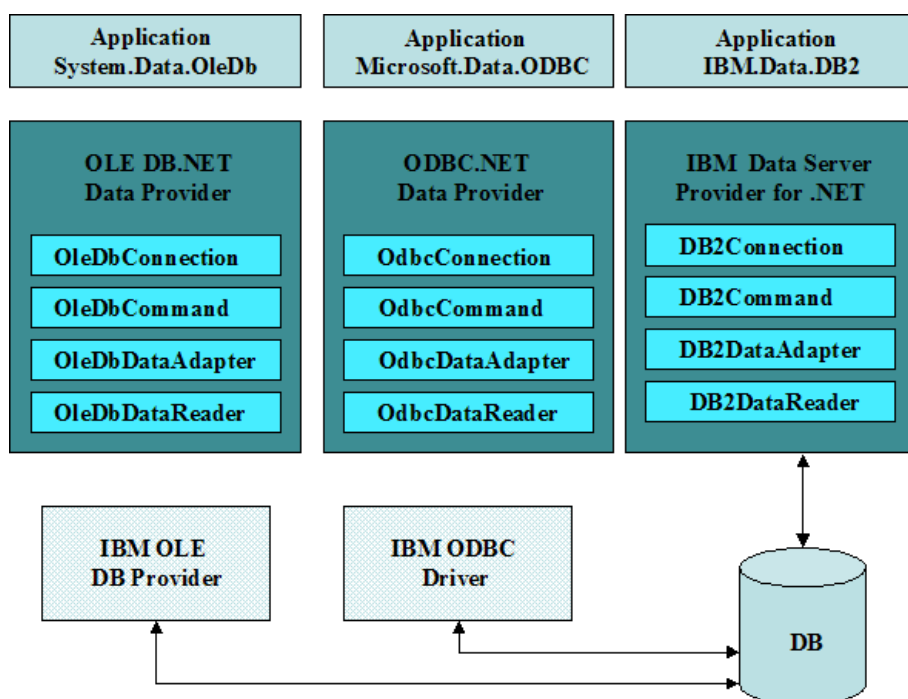
- IBM Data Server Provider for .NET
- OLE DB .NET Data Provider
- ODBC .NET Data Provider

IBM Data Server Provider for .NET is a high performance, managed type ADO.NET data provider, which is provided by IBM and has a much better performance than the OLE DB and ODBC .NET data providers.

OLE DB .NET Data Provider is a bridge provider from Microsoft that passes the ADO.NET request to the native IBM OLE DB provider (IBMDADB2).

ODBC.NET Data Provider is a bridge provider from Microsoft that passes ADO.NET requests to the IBM ODBC Driver.

IBM Data Server Provider for .NET is recommended for any new ADO.NET application development. It provides the best performance since it doesn't require an extra layer or bridge as shown in *Figure 6-3*.



**Figure 6.3 - .NET Data Providers for DB2 applications**

As illustrated in the above *Figure 6.3* each provider delivers the same core functionalities or classes described earlier: Connection, Command, DataAdapter, DataReader.

In the 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.

The **IBM.Data.DB2** namespace is the .NET Framework Data Provider for IBM data servers. The IBM Data Server Provider for .NET extends support for the ADO.NET interface and delivers high-performing, secure access to data.

**Note:**

For more details on namespaces refer to

<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.swg.im.dbc.lient.adonet.ref.doc/doc/IBMDataDB2Namespace.html>

**6.2.1.1 Connection**

The **Connection** object is used to connect to a database and control the transactions in ADO.NET. Each of the three data providers mentioned earlier has their own **Connection** Object (**DB2Connection**, **OleDbConnection**, and **OdbcConnection**). The **Connection** object has a public property **ConnectionString**, which is required for establishing a connection to the database. **ConnectionString** requires the database name and other parameters such as user ID and password. For example:

```
connection.ConnectionString = "Database=Sample";
```

The **ConnectionString** property can be set through the **Connection** object constructor, as shown in *Table 6.2 below*.

| Provider                          | Example                                                                                                                                    |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| IBM Data Server Provider for .NET | <pre>DB2Connection connection =     new DB2Connection("Database=SAMPLE");</pre>                                                            |
| OLE DB .NET Data Provider         | <pre>OleDbConnection connection =     new OleDbConnection("Provider=IBMDADB2;" +     "Data Source=sample;UID=userid;PWD=password;");</pre> |
| ODBC.NET Data Provider            | <pre>OdbcConnection connection = new     OdbcConnection("DSN=sample;UID=userid;PWD=password;");</pre>                                      |

**Table 6.2 - Connection string depending on the data provider used**

*Table 6.3* describes the **Connection** object public methods.

| Method name | Description                                                                                                                                                                                     | Example                       |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------|
| <b>Open</b> | This opens a database connection as specified in a <b>connection_string</b> . Connections can be opened by explicitly calling the <b>Open</b> method on the connection or by implicitly using a | <pre>connection.Open();</pre> |

|                         | DataAdapter.                                                 |                                             |
|-------------------------|--------------------------------------------------------------|---------------------------------------------|
| <b>Close</b>            | This closes the database connection                          | <code>connection.Close();</code>            |
| <b>CreateCommand</b>    | This returns a command object associated with the connection | <code>connection.CreateCommand();</code>    |
| <b>BeginTransaction</b> | This begins the database transaction                         | <code>connection.BeginTransaction();</code> |

**Table 6.3 - Connection object public methods**

### 6.2.1.2 Command

The `Command` object allows execution of any supported SQL statement or stored procedure for a given data `Connection` object. A `Connection` object should be previously created, but it does not need to be opened prior to creating the SQL statements.

The `Command` object can be instantiated as shown in *Table 6.4*.

| Provider                          | Example                                             |
|-----------------------------------|-----------------------------------------------------|
| IBM Data Server Provider for .NET | <code>DB2Command cmd = new DB2Command();</code>     |
| OLE DB .NET Data Provider         | <code>OleDbCommand cmd = new OleDbCommand();</code> |
| ODBC.NET Data Provider            | <code>OdbcCommand cmd = new OdbcCommand();</code>   |

**Table 6.4 – Instantiating the Command object depending on the data provider**

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 statement that retrieves a single value from the database, for example:

```
int count=(int)cmd.ExecuteScalar();
```

**Note:**

Object returned by cmd.ExecuteScalar() should be cast to data type of the underlying database object. The above example is valid for a case where a single value is being retrieved from an int column.

### 6.2.1.3 DataAdapter

The data adapter object populates a disconnected **DataSet** with data and performs updates. It contains four optional commands for SELECT, INSERT, UPDATE, and DELETE. Use this object when you want to load or unload data between a **DataSet** and a database.

The **DataAdapter** object can be instantiated as shown in *Table 6.5*.

| Provider                          | Example                                                         |
|-----------------------------------|-----------------------------------------------------------------|
| IBM Data Server Provider for .NET | <code>DB2DataAdapter adapter = new DB2DataAdapter();</code>     |
| OLE DB .NET Data Provider         | <code>OleDbDataAdapter adapter = new OleDbDataAdapter();</code> |

|                        |                                                             |
|------------------------|-------------------------------------------------------------|
| ODBC.NET Data Provider | <pre>OdbcDataAdapter adapter = new OdbcDataAdapter();</pre> |
|------------------------|-------------------------------------------------------------|

**Table 6.5 – Instantiating the DataAdapter depending on the data provider**

Data adapter object has the following public properties:

- 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** updates 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);
```



#### 6.2.1.4 DataReader

**DataReader** is used 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.

The **DataReader** object can be used as shown in *Table 6.6*.

| Provider                          | Example                                                    |
|-----------------------------------|------------------------------------------------------------|
| IBM Data Server Provider for .NET | <code>Db2DataReader reader = cmd.ExecuteReader();</code>   |
| OLE DB .NET Data Provider         | <code>OleDbDataReader reader = cmd.ExecuteReader();</code> |
| ODBC.NET Data Provider            | <code>OdbcDataReader reader = cmd.ExecuteReader();</code>  |

**Table 6.6 – Using DataReader depending on the data provider used**

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 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 xxxx, for example:

```
Console.WriteLine (reader.GetString(1));
```

### 6.2.2 DataSet for ADO.NET

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 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 the XML representation of data in the `DataSet`, for example:

```
Console.WriteLine(ds.GetXml());
```

- **ReadXML:** This reads the 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" );
```

*Listing 6.1, Listing 6.2, and Listing 6.3* provide an ADO.NET sample C# code snippet that demonstrates the use of the various DB2 Data Providers. Each listing contains the C# code to connect to a database. These sample code snippets require the DB2 *SAMPLE* database to be created.

*Listing 6.1* shows the C# code snippet to connect to a database using the IBM Data Server .NET Data Provider.

```
String connectString = "Database=SAMPLE";  
DB2Connection conn = new DB2Connection(connectString);  
conn.Open();  
return conn;
```

**Listing 6.1 - C# code snippet to connect to a database using the IBM Data Server Provider for .NET**

*Listing 6.2* shows the C# code snippet to connect to a database using the OLE DB .NET Data Provider

```
OleDbConnection con = new OleDbConnection("Provider=IBMDADB2;" +
    "Data Source=sample;UID=userid;PWD=password;");
con.Open();
```

**Listing 6.2 - C# code snippet to connect to a database using the OLE DB .NET Data Provider**

*Listing 6.3* shows the C# code snippet to connect to a database using the ODBC.NET Data Provider

```
OdbcConnection con = new
OdbcConnection("DSN=sample;UID=userid;PWD=password;");
con.Open();
```

**Listing 6.3 - C# code snippet to connect to a database using the ODBC.NET Data Provider**

The code samples can be compiled using the following command:

```
csc NETSamp.cs /r:<DB2 Install Path>\bin\netf20\IBM.Data.DB2.dll
```

where *<DB2 Install Path>* is the path where DB2 is installed.

In C#, all errors are treated as instances of an exception. Error handling in ADO.NET is performed using a `try/catch/finally` block or using the `On Error` construct.

### 6.3 Setting up the environment

The set up required to start developing .NET applications to access DB2 is fairly straight forward. You only need to install either a DB2 client, or a DB2 server which include a .NET data provider. *Table 6.1* shows the the .NET data providers that are shipped with DB2 Version 9.7 clients and servers, and their 32-bit and 64-bit support levels.

| .Net data provider framework                                                          | 32-bit support | 64-bit support |
|---------------------------------------------------------------------------------------|----------------|----------------|
| IBM Data Server Provider for .NET Framework Version 1.1                               | Yes            | No             |
| IBM Data Server Provider for .NET Framework Version 2.0, Version 3.0, and Version 3.5 | Yes            | Yes            |

**Table 6.1 - 32-bit and 64-bit support in IBM Data Server data providers for .NET**

During the installation of the DB2 client or server software, one of these two IBM Data Server Providers for .NET editions will be installed:

- For Windows on 32-bit AMD and Intel systems (x86)

The 32-bit edition of the IBM Data Server Provider for .NET Framework version 2.0, version 3.0 and version 3.5 is installed with DB2 9.7. The IBM Data Server Provider for .NET Framework version 1.1 is also installed.

- For Windows on AMD64 and Intel EM64T systems (x64)

Only the 64-bit edition of the IBM Data Server Provider for .NET is installed with DB2 Version 9.7. The IBM Data Server Provider for .NET, Framework 1.1 is not installed. The 64-bit edition of the IBM Data Server Provider for .NET does not support the IA-64 architecture.

You can run 32-bit .NET applications on a 64-bit Windows instance, using a 32-bit edition of the IBM Data Server Provider for .NET. To get a 32-bit IBM Data Server Provider for .NET on your 64-bit computer, you can install the 32-bit version of IBM Data Server Driver Package.

**Note:**

For more details about the IBM Data Server Provider for .NET for rapid application development, visit the IBM Information Management and Visual Studio .NET zone at <http://www.ibm.com/developerworks/data/zones/vstudio/index.html>.

You can use Visual Studio to develop .NET applications. You can learn more about using the IBM Database Add-Ins for Visual Studio in *section 6.3.1*.

### 6.3.1 IBM Database Add-Ins for Visual Studio

The IBM Database Add-Ins for Visual Studio are a collection of features that integrate seamlessly into your Visual Studio development environment so that you can work with DB2 servers and develop DB2 procedures, functions, and objects.

IBM Database Add-Ins for Visual Studio are designed to present a simple interface to DB2 databases. For example, instead of using SQL, the creation of database objects can be done using designers and wizards. And for situations where you do need to write SQL code, the integrated DB2 SQL editor has the following features:

- Colored SQL text for increased readability
- Integration with the Microsoft® Visual Studio IntelliSense feature, which provides for intelligent auto-completion while you are typing DB2 scripts

With IBM Database Add-Ins for Visual Studio, you can:

- Open various DB2 development and administration tools
- Create and manage DB2 projects in the Solution Explorer
- Access and manage DB2 data connections (in Visual Studio 2005 or later you can do this from the Server Explorer)

- Create and modify DB2 scripts, including scripts to create stored procedures, functions, tables, views, indexes, and triggers

**Note:**

At the time of writing the IBM Database Add-Ins for Visual Studio are not yet supported with Visual Studio 2010.

### 6.3.1.1 Installing the IBM Database Add-Ins for Visual Studio

The IBM Database Add-Ins for Visual Studio is a separately installable component and can be downloaded from <http://www.ibm.com/db2/express/download.html>. After you install a DB2 product, install the IBM Database Add-Ins for Visual Studio by double clicking on the executable `db2exc_vsai_XXX_WIN_x86.exe`, where `XXX` represents a version number that matches the version number of the DB2 server. The installation of the add-ins is straight forward; simply take all defaults. *Figure 6.4, 6.5 and 6.6* illustrate some of the panels that will appear when you install the IBM Database Add-Ins for Visual Studio.

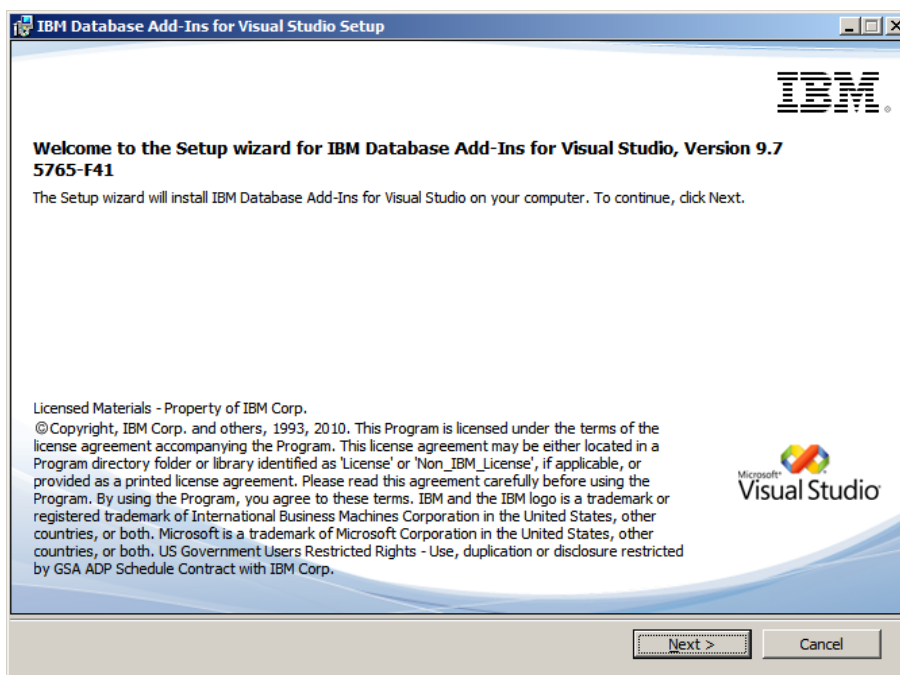


Figure 6.4 - Installing the IBM Database Add-Ins for Visual Studio - Welcome panel

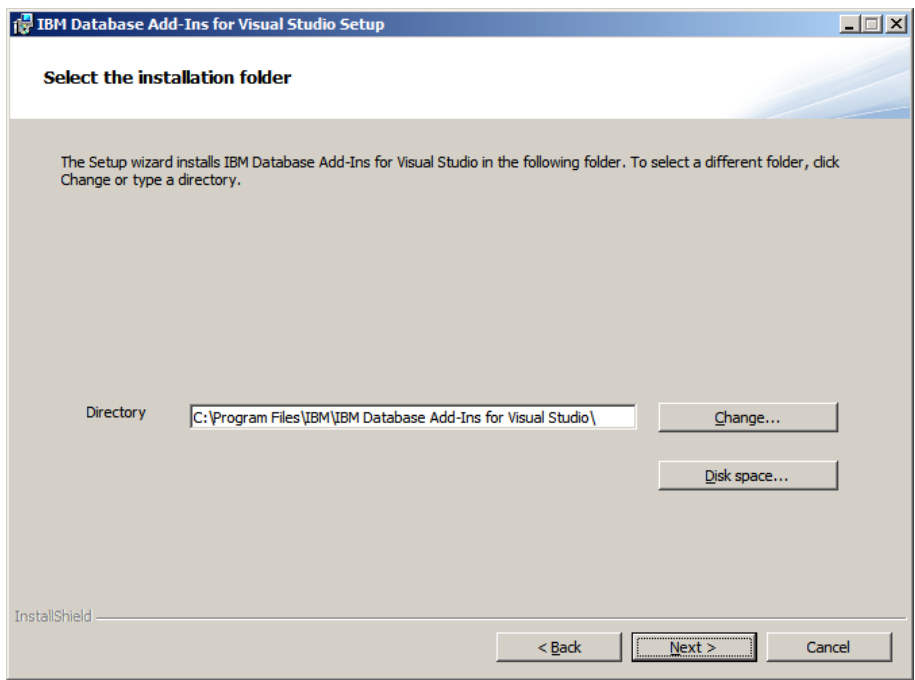


Figure 6.5 - Installing the IBM Database Add-Ins for Visual Studio - Select the installation folder

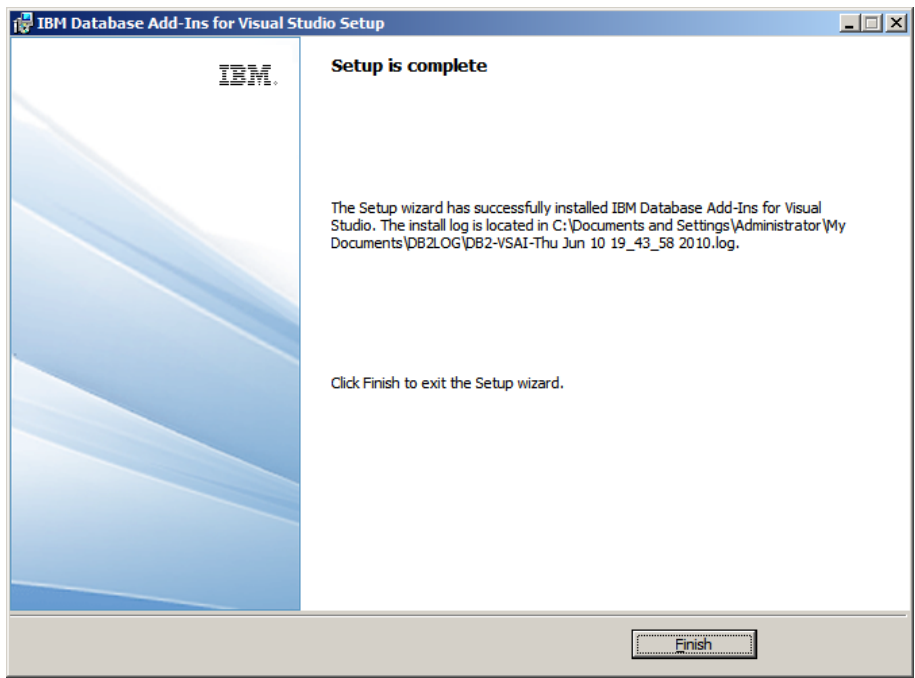


Figure 6.6 - Installing the IBM Database Add-Ins for Visual Studio - Setup complete

If you do not have Visual Studio installed on your computer, the add-ins will not install.

**Note:**

For more details about using the IBM Database Add-Ins for Visual Studio, visit the IBM Information Management and Visual Studio zone at <http://www.ibm.com/developerworks/data/zones/vstudio/index.html>.

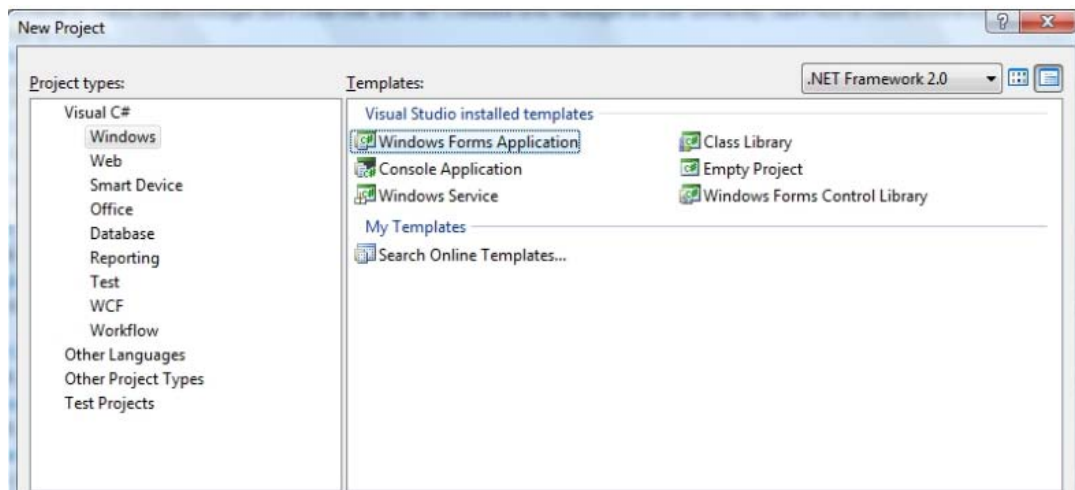
### 6.3.2 Using Visual Studio with DB2

In this section, we describe the steps to create a Visual C# ADO.NET application. You can use almost the same steps to create an application in Visual Basic, Visual J#, Visual C/C#, and so on.

#### 6.3.2.1 Creating a Visual C# ADO.NET application

The following steps create a Visual C# ADO.NET application using Visual Studio

1. Create a Visual C# project: *File->New->Project->VC# -> Windows Application*
  - On the *File* menu, select *New -> Project*
  - On the *New Project* dialog, for *Project Types* select the *Visual C# Projects* folder. Select *Windows Forms Application* project template. Specify the project name and directory under *Name* and *Location* respectively. The Solution Explorer window should look similar to *Figure 6.7*.



**Figure 6.7 - Creating a new Visual C# .NET project**

2. Once the installation of the IBM Database Add-ins for Visual Studio is complete, the data source tab will appear besides the Toolbox tab. This is illustrated in *Figure 6.8*.

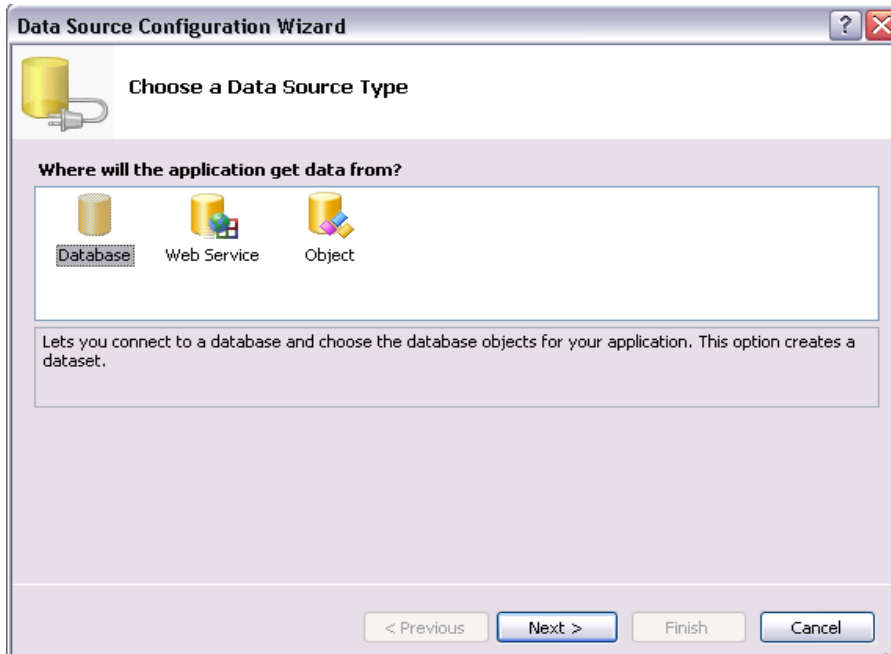


**Figure 6.8 – Data Source tab appears on the bottom left corner after registration**

3. Drag and drop the data adaptor on the Windows form. Also drag and drop the data grid control on the form. Change the name property according to the requirement.
4. Click on the *db2DataAdapter* and then select the *Generate DataSet* option. Select *New: DB2DataSet1* and click OK. This will create a dataset object and an instance of that object named *db2DataSet11*.
5. Click on *dataGrid\_employee* on the form and in the properties window make the following property selections, in order to data-bind the EMPLOYEE table to the data grid:
  - For DataSource - Select Db2DataSet111
  - For DataMember - Select EMPLOYEE

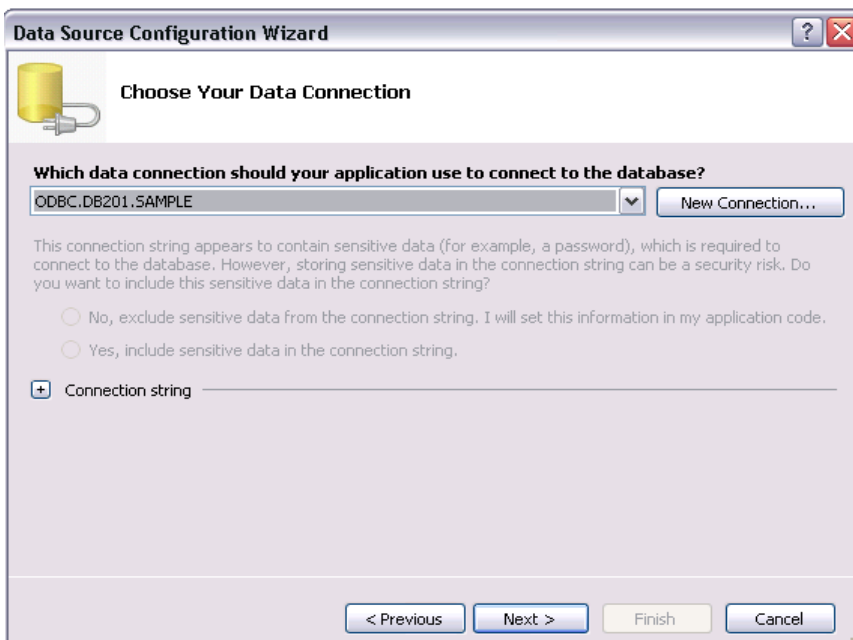


This is illustrated in *Figure 6.9*.



**Figure 6.9 – Selecting the data source**

6. Generate a connection string by using the GUI tool as is illustrated in *Figure 6.10*

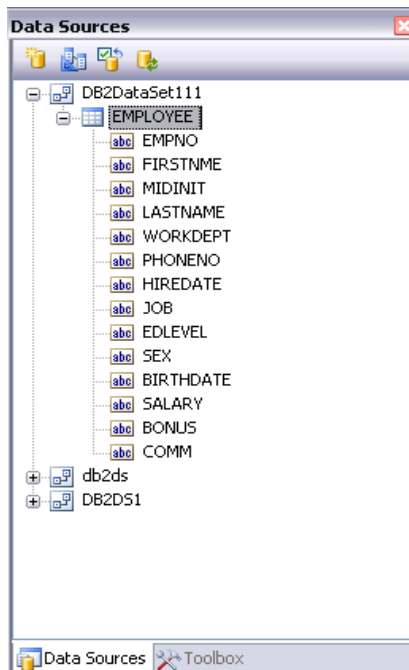


**Figure 6.10 – Forming the Connection string**

The connection string can be formed at runtime as follows:

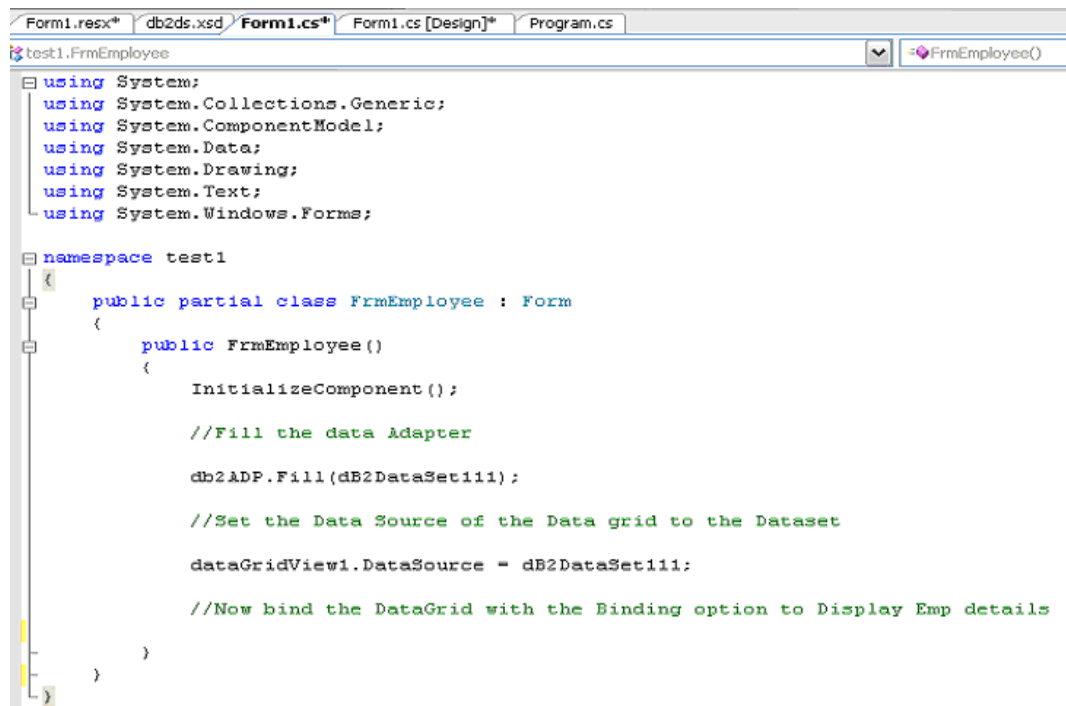
```
DB2Connection con = new DB2Connection("Database=sample;User  
ID=db2admin;Server=localhost:50000;Persist Security  
Info=True;Password=db2admin");  
  
    DB2DataAdapter da = new DB2DataAdapter("select * from  
administrator.Employees", con);  
  
    DataSet db2DataSet11 = new DataSet();  
  
    da.Fill(db2DataSet11);  
  
    GridView1.DataSource = db2DataSet11;  
  
    GridView1.DataBind();
```

7. After successfully editing the data source the Data Member can be listed in the data sources tab as illustrated in *Figure 6.11*



**Figure 6.11 – Listing Data Member in data sources tab**

8. Double click on *form.cs*; it will present a code window. Write a small program in it as illustrated in *Figure 6.12* below.



```

Form1.resx* db2ds.xsd Form1.cs* Form1.cs [Design]* Program.cs
test1.FrmEmployee FrmEmployee()
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace test1
{
    public partial class FrmEmployee : Form
    {
        public FrmEmployee()
        {
            InitializeComponent();

            //Fill the data Adapter

            db2ADP.Fill(db2DataSet111);

            //Set the Data Source of the Data grid to the Dataset

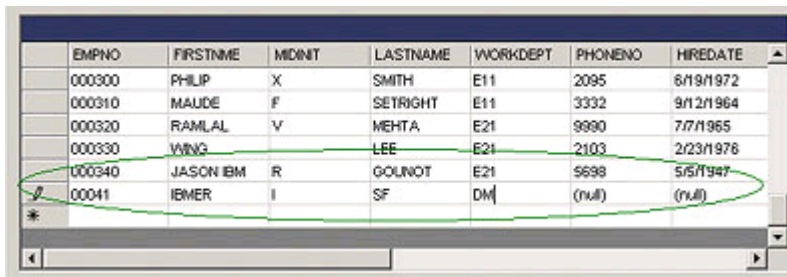
            dataGridView1.DataSource = db2DataSet111;

            //Now bind the DataGrid with the Binding option to Display Emp details
        }
    }
}

```

Figure 6.12 – Writing code in form.cs

9. To test this sample program, click on *Debug -> Start Without Debugging* from the Visual Studio .NET menu. This will build and run the application. If everything went well, you should be able to see the table data in the grid retrieving data from the DB2 SAMPLE database as shown in *Figure 6.13* below.



| EMPNO  | FIRSTNAME | MIDINIT | LASTNAME | WORKDEPT | PHONENO | HIREDATE  |
|--------|-----------|---------|----------|----------|---------|-----------|
| 000300 | PHILIP    | X       | SMITH    | E11      | 2095    | 6/19/1972 |
| 000310 | MAUDE     | F       | SETRIGHT | E11      | 3332    | 9/12/1964 |
| 000320 | RAMLAL    | V       | MEHTA    | E21      | 9990    | 7/7/1965  |
| 000330 | WING      |         | LEE      | E21      | 2103    | 2/23/1976 |
| 000340 | JASON IBM | R       | GOLINOT  | E21      | 5698    | 5/5/1947  |
| 00041  | IBMER     | I       | SF       | DM       | (null)  | (null)    |

Figure 6.13 – Retrieved values in Data Grid.

## 6.4 Developing .NET - DB2 applications

In this section, you will learn how to establish connections to a DB2 database and retrieve some information. We will provide examples using each of the data providers discussed earlier, and we use the *SAMPLE* database, with *db2admin* as the username and *mypsw* as the password. The sample code is provided as part of the file

Exercise\_Files\_DB2\_Application\_Development.zip provided with this book.  
Follow four steps to connect to DB2 using any .NET Data Provider and run the programs:

### Step 1: Configure connectivity to the database.

Set up connectivity to the DB2 database as discussed in the *DB2 Client Connectivity* chapter of the eBook [Getting started with DB2 Express-C - 3<sup>rd</sup> Edition](#).

### Step 2: Set the reference

To set the reference, go to menu bar and select *view -> solution explorer*. On your project right click and select *References*. In *References* select the data provider with which you intend to work. This is illustrated in *Figure 6.14*.

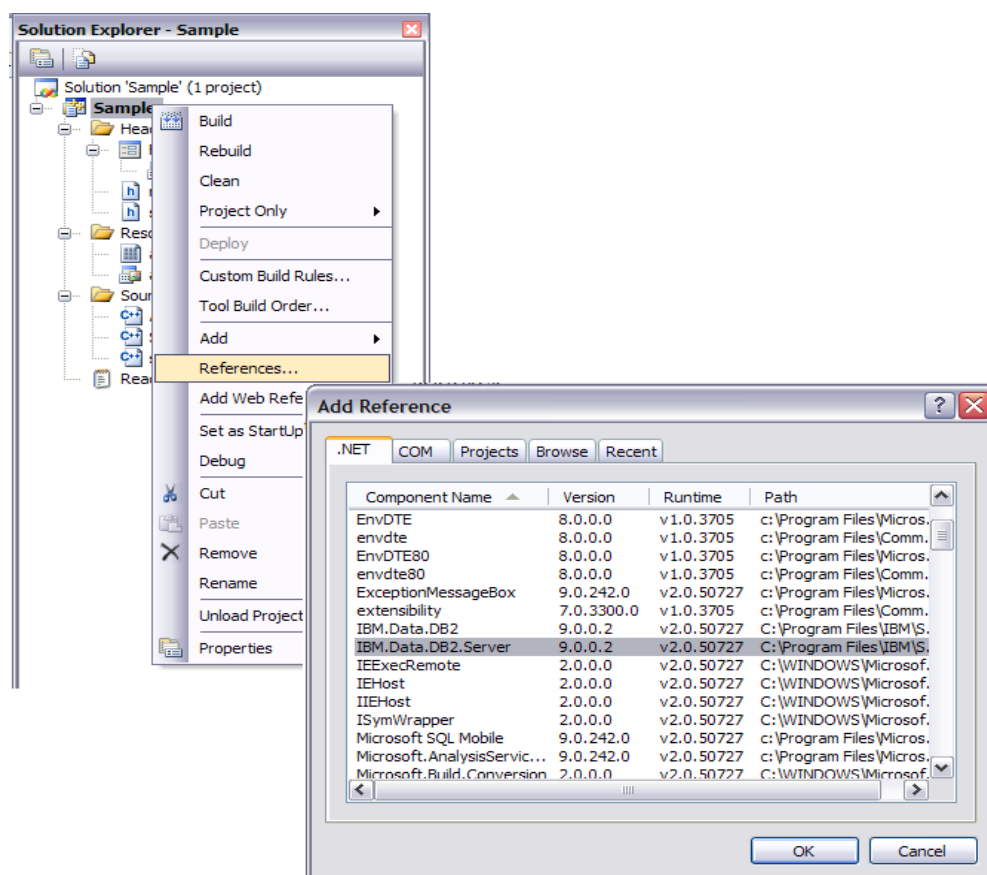
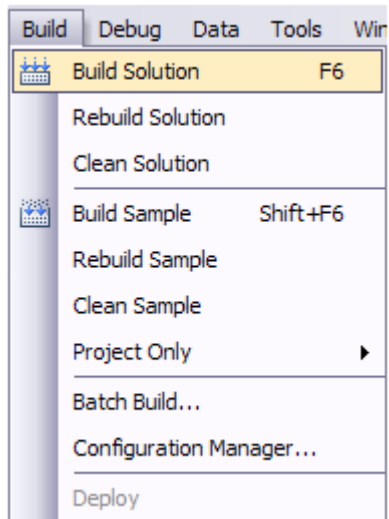


Figure 6.14 – Adding the Data Provider as Reference

### Step 3: Compile the programs

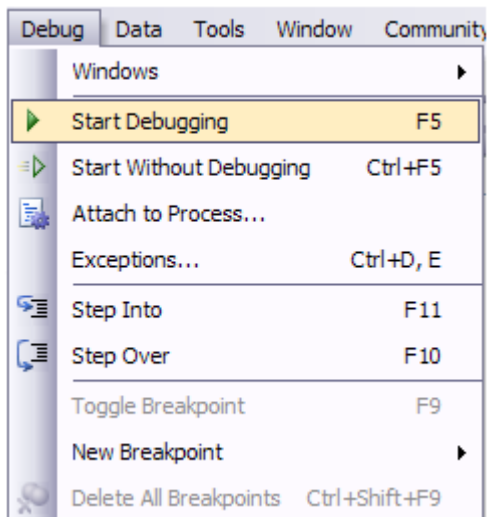
To compile the program, on the menu bar select *Build* -> *Build Solution* option. This is illustrated in *Figure 6.15*.



**Figure 6.15 – Compiling a program**

#### Step 4: Execute the program

To execute the program, on the menu bar select *Debug* -> *Start Debugging* option as shown in *Figure 6.16*.



**Figure 6.16 – Executing a program**

### 6.4.1 Connecting to a DB2 database with the IBM Data Server Provider for .NET

*Listing 6.4* shows how to connect to a DB2 database in Visual Basic using the IBM Data Server Provider for .NET and issuing a simple SELECT statement.

```
(1) Imports IBM.Data.DB2

Module Module1
    Sub Main()
        Dim cmd As DB2Command
        Dim con As DB2Connection
        Dim rdr As DB2DataReader
        Dim v_IBMREQD As String
        Try
            con = New DB2Connection("Database=sample;" +
                                   "UID=db2admin;PWD=mypsw;")
            cmd = New DB2Command()
            (2) cmd.Connection = con
            (3) cmd.CommandText = "SELECT * FROM SYSIBM.SYSDUMMY1"
            cmd.CommandTimeout = 20
            con.Open()
            (4) rdr = cmd.ExecuteReader(CommandBehavior.SingleResult)
            v_IBMREQD = ""
            While (rdr.Read() = True)
                v_IBMREQD = rdr.GetString(0)
            End While
            Dim strMsg As String
            strMsg = "Successful retrieval of record. Column " +
                    "'IBMREQD' has a value of '" + v_IBMREQD + "'"
            Console.WriteLine(strMsg)
            Console.ReadLine()
            (5) rdr.Close()
            (6) con.Close()
        Catch myException As DB2Exception
        End Try
    End Sub
End Module
```

#### Listing 6.4 - Visual Basic ADO.NET code snippet (IBM Data Server Provider for .NET)

Let's review each of the items shown in *Listing 6.4*:

1. This statement imports the IBM.Data.DB2, which indicates the use of IBM Data Server Provider for .NET.

2. The connection string is assigned to the `cmd.Connection`, to be able to execute the query later.
3. The `cmd.CommandText` sets the query to be executed at the data source with the help of connection established before.
4. The `rdr` is an instance of the `DataReader` object and executes the command using the `cmd.ExecuteReader` method.
5. Close the `DataReader` object.
6. Close the `Connection` object.

*Listing 6.5* shows how to connect to a DB2 database in C# using the IBM Data Server Provider for .NET and issuing a simple SELECT statement.

```
using System;
using System.Collections.Generic;
using System.Text;
using IBM.Data.DB2;

namespace c1
{
    class Program
    {
        static void Main(string[] args)
        {
            DB2Command cmd = null;
            DB2Connection con = null;
            DB2DataReader rdr = null;
            string v_IBMREQD;
            int rowCount;
        try{
            con = new DB2Connection("Database=sample;UID=db2admin;PWD=mypsw;");
            cmd = new DB2Command();
            cmd.Connection = con;
            cmd.CommandText = "SELECT * FROM SYSIBM.SYSDUMMY1";
            cmd.CommandTimeout = 20;
            con.Open();

            rdr = cmd.ExecuteReader(System.Data.CommandBehavior.SingleResult);
            v_IBMREQD = "";
            while (rdr.Read() == true) {
                v_IBMREQD = rdr.GetString(0); }
            string strMsg;
            strMsg = " Successful retrieval of record. Column" +
```

```
        " 'IBMREQD' has a value of '" + v_IBMREQD + "'";
        Console.WriteLine(strMsg);

        Console.Read();
        rdr.Close();
        con.Close();
    } catch (DB2Exception myException) { }
    }
    private static void getch()
    {
        throw new Exception("The method or operation is not implemented.");
    }
}
}
```

**Listing 6.5 - C# ADO.NET code snippet (IBM Data Server .NET Data Provider)**

#### 6.4.2 Connecting to a DB2 database with the OLE DB .NET Data Provider

*Listing 6.6* shows how to connect to a DB2 database in Visual Basic using the OLE DB .NET Data Provider and issuing a simple SELECT statement.

```
Imports System.Data.OleDb
Module Module1
    Sub Main()
        Dim cmd As OleDbCommand
        Dim con As OleDbConnection
        Dim rdr As OleDbDataReader
        Dim v_IBMREQD As String
        Try
            con = New OleDbConnection("DSN=sample;UID=db2admin;PWD=myspw;"
+ "Provider='IBMDADB2';")
            cmd = New OleDbCommand()
            cmd.Connection = con
            cmd.CommandText = "SELECT * FROM SYSIBM.SYSDUMMY1"
            cmd.CommandTimeout = 20
            con.Open()
            rdr = cmd.ExecuteReader(CommandBehavior.SingleResult)
            While rdr.Read()
                v_IBMREQD = rdr.GetString(0)
            End While
            Dim str1 As String
            str1 = "'IBMREQD' has a value of '" + v_IBMREQD + "'
'Console.WriteLine('IBMREQD' has a value of '" +
                v_IBMREQD + "'")
        End Try
    End Sub
End Module
```



```

        Console.WriteLine(str1)
        Console.ReadLine()
        rdr.Close()
        con.Close()
    Catch myException As OleDbException
    End Try
End Sub
End Module

```

### Listing 6.6 - Visual Basic ADO.NET code snippet (OLE DB .NET Data Provider)

*Listing 6.7* shows how to connect to a DB2 database in C# using the OLE DB .NET Data Provider and issuing a simple SELECT statement.

```

using System;
using System.Collections.Generic;
using System.Text;
using System.Data.OleDb;

namespace c2
{
    class Program
    {
        static void Main(string[] args)
        {
            OleDbCommand cmd = null;
            OleDbConnection con = null;
            OleDbDataReader rdr = null;
            int rowCount;
            string v_IBMREQD, strMsg;
            try
            {
                con = new OleDbConnection("DSN=sample;UID=db2admin;PWD=mypsw;" +
"Provider='IBMDADB2';");
                cmd = new OleDbCommand();
                cmd.Connection = con;
                cmd.CommandText = "SELECT * FROM SYSIBM.SYSDUMMY1";
                cmd.CommandTimeout = 20;
                con.Open();
                rdr
cmd.ExecuteReader(System.Data.CommandBehavior.SingleResult);
                v_IBMREQD = "";
                while (rdr.Read() == true) {
                    v_IBMREQD = rdr.GetString(0); }
                strMsg = " Successful retrieval of record. Column" + "
                    'IBMREQD' has a value of '" + v_IBMREQD + "'";
            }
            catch { }
        }
    }
}

```

```
    Console.WriteLine(strMsg);
    Console.ReadLine();
    rdr.Close();
    con.Close();
} catch (OleDbException myException) { }
    }
}
```

**Listing 6.7 - C# ADO.NET code snippet (OLE DB .NET Data Provider)**

### 6.4.3 Connecting to a DB2 database using the ODBC .NET Data Provider

*Listing 6.8* shows how to connect to a DB2 database in Visual Basic using the ODBC .NET Data Provider and issuing a simple SELECT statement.

```
Imports System.Data.Odbc
Module Module1
    Sub Main()
        Dim cmd As OdbcCommand
        Dim con As OdbcConnection
        Dim rdr As OdbcDataReader
        Dim v_IBMREQD As String
        Try
            con = New OdbcConnection("DSN=sample;UID=db2admin;PWD=mypsw;"
+ "Driver={IBM DB2 ODBC DRIVER};")
            cmd = New OdbcCommand()
            cmd.Connection = con
            cmd.CommandText = "SELECT * FROM SYSIBM.SYSDUMMY1"
            cmd.CommandTimeout = 20
            con.Open()

            rdr = cmd.ExecuteReader(CommandBehavior.SingleResult)
            While rdr.Read()
                v_IBMREQD = rdr.GetString(0)
            End While
            Dim str1 As String
            str1 = "'IBMREQD' has a value of '" + v_IBMREQD + "'"
            Console.WriteLine(str1)
            Console.ReadLine()
            rdr.Close()
            con.Close()
        Catch myException As OdbcException
```

```

        End Try
    End Sub
End Module

```

### Listing 6.8 - Visual Basic ADO.NET code snippet (ODBC .NET Data Provider)

*Listing 6.9* shows how to connect to a DB2 database in C# using the ODBC .NET Data Provider and issuing a simple SELECT statement.

```

using System;
using System.Collections.Generic;
using System.Text;
using System.Data.Odbc;

namespace c3
{
    class Program
    {
        static void Main(string[] args)
        {
            OdbcCommand cmd = null;
            OdbcConnection con = null;
            OdbcDataReader rdr = null;
            string v_IBMREQD, strMsg;
            try
            {
                con = new OdbcConnection("DSN=sample;UID=db2admin;PWD=mypsw;"
+ "Driver={IBM DB2 ODBC DRIVER};");
                cmd = new OdbcCommand();
                cmd.Connection = con;
                cmd.CommandText = "SELECT * FROM SYSIBM.SYSDUMMY1";
                cmd.CommandTimeout = 20;
                con.Open();
                rdr =
cmd.ExecuteReader(System.Data.CommandBehavior.SingleResult);
                v_IBMREQD = "";
                while (rdr.Read() == true) {
                    v_IBMREQD = rdr.GetString(0); }
                strMsg = " Successful retrieval of record. Column" +
                    " 'IBMREQD' has a value of '" + v_IBMREQD + "'";
                Console.WriteLine(strMsg);
                Console.ReadLine();
                rdr.Close();
                con.Close();
            } catch (OdbcException myException) { }
        }
    }
}

```

```
}  
}
```

### Listing 6.9 - C# ADO.NET code snippet (ODBC .NET Data Provider)

There is an extra step needed in Step 1, previously described, when using the ODBC .NET Data Provider:

#### Step 1: Configuring connectivity

- The DB2 database must be cataloged as an ODBC data source. To catalog an ODBC data source follow the steps given below:

```
db2 catalog system ODBC data source <databasename>  
or  
db2 catalog user ODBC data source <databasename>
```

- To list ODBC data sources:

```
db2 list system ODBC data sources  
or  
db2 list user ODBC data sources
```

Compiling and executing steps are the same as listed before.

## 6.5 Data Manipulation using .NET

The following C# sample code snippets illustrate the methods to SELECT, INSERT, UPDATE and DELETE using the IBM Data Server Provider for .NET classes -- **DB2Connection**, **DB2Command**, and **DB2Transaction**.

*Listing 6.10* illustrates how to perform a SELECT.

```
(1) cmd.CommandText = "SELECT deptnumb, location " +  
    " FROM org " +  
    " WHERE deptnumb < 25";  
(2) DB2DataReader reader = cmd.ExecuteReader();
```

### Listing 6.10 - C# code demonstrating a SELECT

Let's review each of the items shown in *Listing 6.10*:

- (1) The **cmd.CommandText** specifies the query to be executed using the connection established before.
- (2) The **reader** is an instance of the **DataReader** object and executes the command using the **cmd.ExecuteReader** method. The **ExecuteReader** method sends the **CommandText** to the **Connection** and builds a **DB2DataReader**.

*Listing 6.11* illustrates how to perform an INSERT.

```
// Use the INSERT statement to insert data into the 'staff' table.
cmd.CommandText = "INSERT INTO staff(id, name, dept, job, salary) " +
"    VALUES(380,      'Pearce',      38,      'Clerk',      13217.50)," +
"    (390, 'Hachey', 38, 'Mgr', 21270.00), " +
"    (400, 'Wagland', 38, 'Clerk', 14575.00)";
(1) cmd.ExecuteNonQuery();
```

#### Listing 6.11 - C# code demonstrating an INSERT

Let's review the item shown in *Listing 6.11*:

- (1) The `cmd.ExecuteNonQuery` executes an SQL statement against the connection and returns the number of rows affected.

*Listing 6.12* illustrates how to perform an UPDATE.

```
// This method demonstrates how to update rows in a table
cmd.CommandText = "UPDATE staff " +
"    SET salary = (SELECT MIN(salary) " +
"    FROM staff " +
"    WHERE id >= 310) " +
"    WHERE id = 310";
cmd.ExecuteNonQuery();
```

#### Listing 6.12 - C# code demonstrating an UPDATE

*Listing 6.13* illustrates how to perform DELETES.

```
// This method demonstrates how to delete rows from a table

{
    try
        cmd.CommandText = "DELETE FROM staff " +
"    WHERE id >= 310 " +
"    AND salary > 20000";
        cmd.ExecuteNonQuery();
    }
}
```

#### Listing 6.13 - C# code demonstrating a DELETE

### 6.5.1 Building and Running the sample program

The code snippets shown above were taken from the program `TbUse.cs` which is part of the sample programs provided with DB2 and can be found under the directory `C:\Program Files\IBM\SQLLIB\samples\.NET\cs`.

To run the program follow these steps:

1. Compile the `TbUse.cs` file with `bldapp.bat` from a DB2 Command Window as follows:

```
bldapp TbUse
```

or compile `TbUse.cs` with the makefile by entering the following at the command prompt:

```
nmake TbUse
```

2. Run the `TbUse` program by entering the program name at the command prompt:

```
TbUse
```

**Note:**

For more information about sample programs, refer to:

<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.samp.top/doc/doc/c0007609.html>

## 6.6 Exercises

Write a small C# program to access BLOB data in the **SAMPLE** database. Follow these steps:

1. Run the following command from a DB2 Command Window to create the **SAMPLE** database if it was not already created: `db2samp1`
2. Write a C# program to update and retrieve BLOB and CLOB data from the `EMP_PHOTO` and `EMP_RESUME` tables. If you find this exercise difficult to do, the sample program for this exercise is provided in the project `db2lobs.zip`, under the "Retrieving and updating BLOBs and CLOBs " section in the following link : <http://www.ibm.com/developerworks/data/library/techarticle/0304surange/0304surange.html#resources>
3. Modify the schema name in the program appropriately and test it.
4. To Run the C# program, follow the steps listed in section 6.6.1.

## 6.7 Summary

In this chapter, you have learned how to set up the environment to develop .NET applications using DB2. You learned the various types of data providers and the way you need to code depending on the provider chosen. The IBM Data Server data provider for .NET is the recommended one because it is best for performance. You also learned about

the IBM Database Add-ins for Visual Studio. The chapter also discussed how to connect to a DB2 database, and how to manipulate DB2 data using sample programs.

### 6.8 Review questions

1. What is the difference between the FieldCount and HasRows public properties of the DataReader object?
2. What configuration do you need to do differently when using the ODBC .NET Data Provider?
3. Name and explain the two components on which the Data Access in ADO.NET relies on.
4. Can you use a 32-bit IBM Data Server Provider for .NET on a 64-bit computer?
5. How can you use the Visual Studio Add-Ins with DB2?
6. The key component of disconnected data access in ADO.NET is:
  - A. DataSet
  - B. DataReader
  - C. Connection
  - D. DataAdapter
  - E. None of the above
7. Which of the following component classes populates a disconnected DataSet with data?
  - A. Command object
  - B. DataReader object
  - C. DataAdapter object
  - D. Dataset
  - E. None of the above
8. Which public property of the Command object is used to describe whether an SQL statement or a stored procedure will be executed?
  - A. CommandText
  - B. CreateParameter
  - C. ExecuteNonQuery
  - D. CommandType
  - E. None of the above

9. All data providers listed below are the .NET data providers for DB2 applications to access a database EXCEPT?
- A. IBM Data Server Provider for .NET
  - B. Microsoft .NET Data Server Provider
  - C. OLE DB .NET Data Provider
  - D. ODBC .NET Data Provider
  - E. None of the above
10. In IBM Database Add-Ins for Visual Studio, what is used to create and manage DB2 Projects?
- A. Object Browser
  - B. Server Explorer
  - C. Solution Explorer
  - D. Device Emulator Manager
  - E. None of the above



# 7

## Chapter 7 - Application development with Ruby on Rails

**Ruby** is an open source, free, dynamic programming language. **Ruby on Rails (RoR)**, or **Rails** for short is a framework built using Ruby. RoR is based on the **Model, View and Controller (MVC)** architecture. By supporting agile development techniques it has enabled many developers to radically reduce the time and pain involved in creating Web applications.

In this chapter you will learn about:

- Ruby on Rails and the MVC architecture
- How to setup Ruby on Rails to work with DB2
- A basic understanding of ways to program RoR applications with DB2
- How to build a simple Ruby on Rails application with DB2

### 7.1 Ruby on Rails applications with DB2: The big picture

Ruby on Rails is organized around the Model, View, and Controller architecture, usually just called MVC. MVC benefits include:

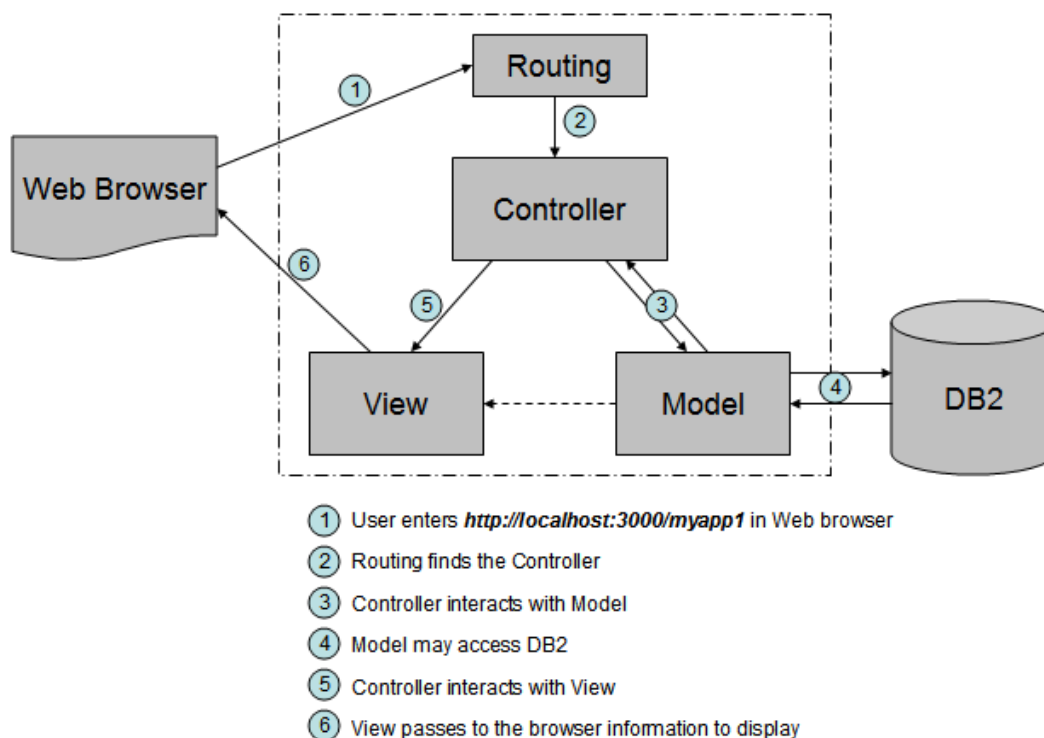
- Isolation of business logic from the user interface
- Making it clear where different types of code belong for easier maintenance

A **Model** represents the information (data) of the application and the rules to manipulate that data. In the case of Rails, models are primarily used for managing the rules of interaction with a corresponding database table. In most cases, one table in your database will correspond to one model in your application. The bulk of your application's business logic will be concentrated in the model. Under this architecture, DB2 provides data persistence.

**Controllers** provide the “glue” between models and views. In Rails, controllers are responsible for processing the incoming requests from the Web browser, interrogating the models for data, and passing that data on to the views for presentation.

**Views** represent the user interface of your application. In Rails, views are often HTML files with embedded Ruby code that performs tasks related solely to the presentation of the data. Views handle the job of providing data to the Web browser or other tool that is used to make requests from your application.

Figure 7.1 illustrates this architecture.



**Figure 7.1 RoR - DB2 applications: The MVC architecture**

In the above figure, a user invokes the application by starting a Web browser and entering the application's URL as shown in (1). In this particular example, we are using RoR's Web server called WEBrick which is installed on the same server (localhost) and listens to port 3000 by default. The URL to invoke the application would look like <http://localhost:3000/myapp1>, where `myapp1` represents the name of the controller.

The incoming request is serviced by the routing module which redirects it to controllers and actions as shown in (2). Then the *Controller* interacts with the *Model* which processes the request as shown in (3). The Model may interact with the DB2 database if access to persistent data is required as shown in (4). The Model reports back to the Controller and the Controller interacts with the View (5). Finally, the View passes to the browser information to display (6).

Since RoR applications are object-oriented applications while most commercial databases, including DB2, use the relational database model, it is common to use **object-relational mapping (ORM)** libraries for these two models to interact. **Active Record** is the ORM

layer supplied with Rails. It closely follows the standard ORM model: tables map to classes, rows to objects, and columns to object attributes.

ORM makes it easier to manage database schema changes. For example, say you built an application which among other things displays information of a table. Your customer asks you to add a couple of columns, and display them. Though this may seem like a simple request, for a large application, this can be quite time consuming, and a source of errors. You would need to look for all applicable SQL statements that were accessing this table and modify them so they include the new columns. This change request could be handled much more easily with **ORM**, because the mapping between entities in the database and entities in the program is included in XML configuration files. Programmers can now conveniently change the XML document to add the columns, and the ORM takes care of generating the correct SQL for you. This may also be useful to avoid code duplication of using the same SQL in different locations in your application. In Active Record (Rails' ORM); however, there are no XML configuration files to change.

Rails philosophy is founded on two main principals:

- a) "Do not repeat yourself" (DRY)
- b) "Convention Over Configuration" (CoC)

The DRY principle is based on the idea that every piece of knowledge in a system should be expressed in one place only. This helps programmers avoid changes in many different parts of the application.

The CoC principle allows programmers to be consistent and reduce the amount of code simply by following some conventions. *Table 7.1* lists some of these conventions and provide examples.

| Convention                                                                                                                                                                                | Example                                                                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| Table names have all lowercase letters and need to be plural                                                                                                                              | books                                                                                                                          |
| The model name uses the class naming convention of unbroken MixedCase and is always the singular of the table name. In the Ruby language the first letter of a class starts in upper case | Book<br>Rails looks for the class definition in a file called book.rb in the /app/models directory.                            |
| Controller class names are pluralized                                                                                                                                                     | BooksController<br>Rails looks for the class definition in a file called books_controller.rb in the /app/controllers directory |

**Table 7.1 - Examples of the CoC principle**

## 7.2 Setting up the RoR environment

Before you can develop RoR applications, you first have to install Ruby, Rails, the drivers and adapters that work with DB2, and configure some files. Depending on your platform, you may have to follow different procedures.

### Note:

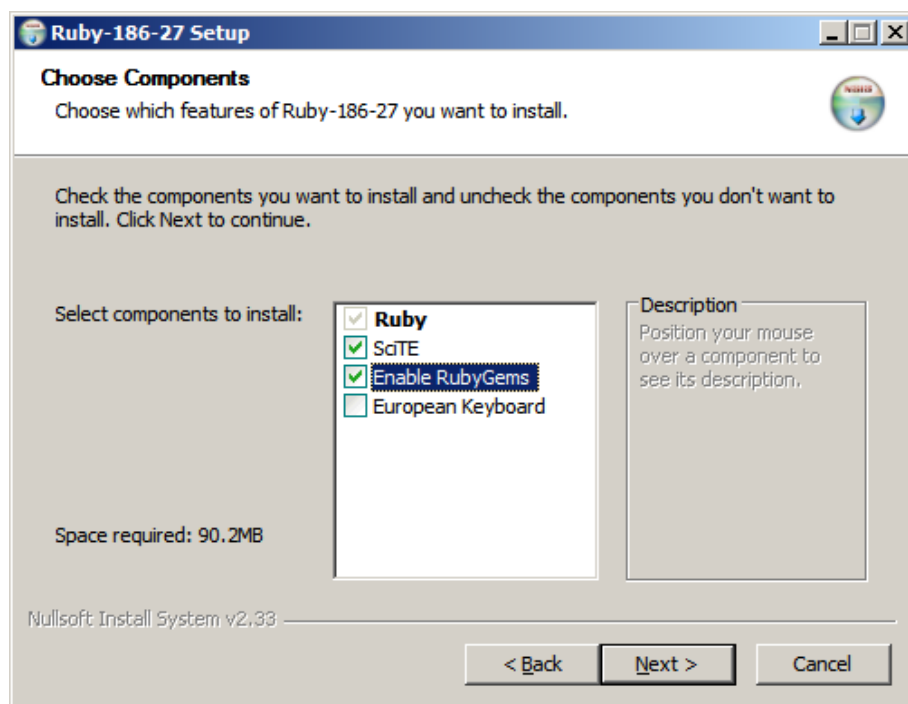
In this book we show you how to set up the RoR environment on Windows. For the setup required on other platforms, refer to the free eBook [Getting started with Ruby on Rails](#), which is part of this DB2 on Campus free book series.

### 7.2.1 Installing Ruby

To install Ruby on Windows, install the code by using the *One-click Installer - Windows* from [http://rubyforge.org/frs/?group\\_id=167](http://rubyforge.org/frs/?group_id=167)

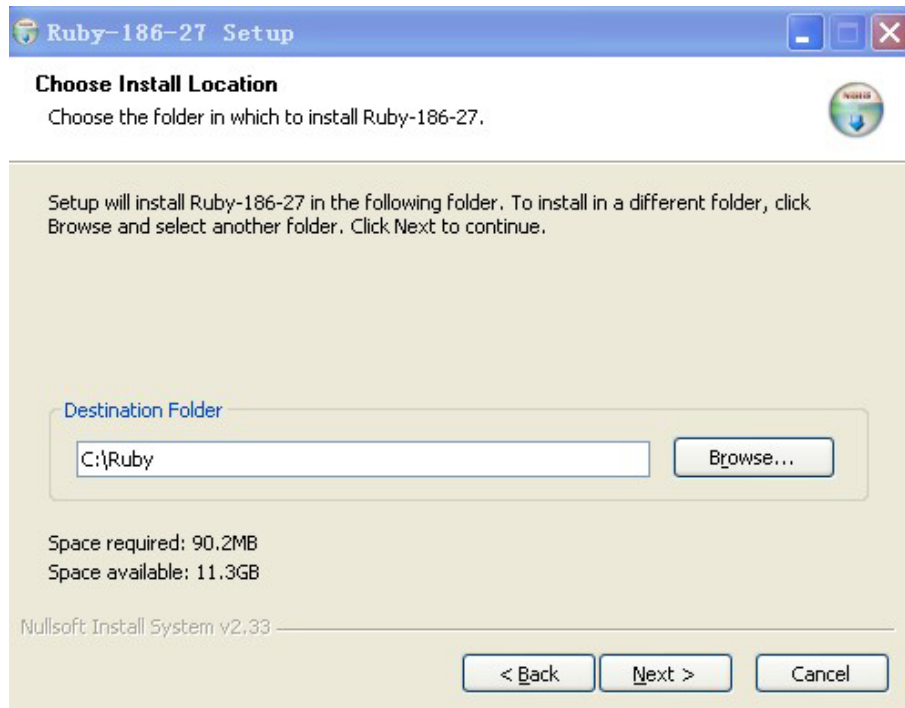
At the time of writing the latest Ruby version is Ruby 1.8.6-27 Release Candidate 2; therefore, we will choose the *ruby186-27\_rc2.exe* executable file. When you click this file, you can either run it, or save it. In our case, we choose *Run* which would download the file to a temporary location in our computer, and then invoke it.

The installation is very simple, all you need to do is accept the license, choose the installing location, and take all the default settings by clicking the *Next* button several times. When you click the *Install* button the installation will start. *Figure 7.2* to *Figure 7.4* show you some of the installation panels.

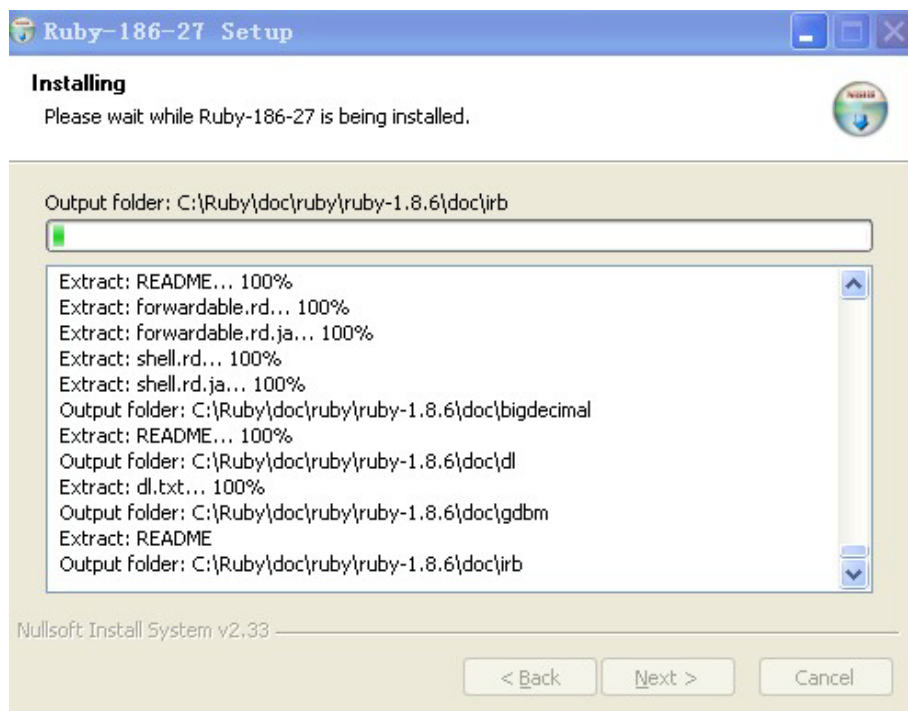


**Figure 7.2 - Ruby installation - Choose components**

For the Choose components panel, we did choose to enable RubyGems. RubyGems is discussed in a later section.



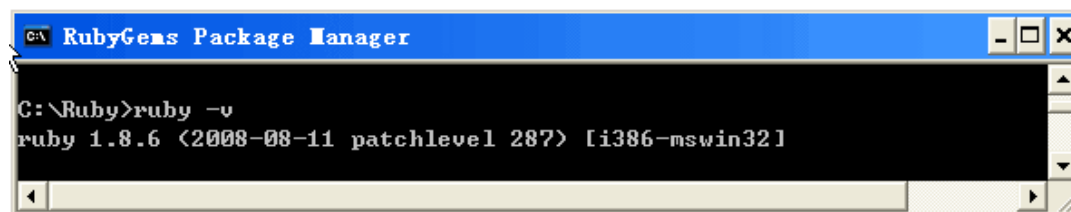
**Figure 7.3 - Ruby installation - Choose the install location.**



**Figure 7.4 - Ruby Installation - Installation progress bar**

After you have installed Ruby, click *Next*, and then *Finish* buttons. If you want to review the readme file, you can check the *Show Readme* checkbox.

Next, it's a good idea to test whether we have installed Ruby successfully by verifying the version installed. From a Windows Command Prompt, you can run the command `ruby` with the `-v` flag to verify the Ruby version. This is shown in *Figure 7.5*.



**Figure 7.5 - Verifying the Ruby version installed**

You can also verify you have the Ruby menus by clicking on *Start -> All Programs -> Ruby-186-27*.

**Note:**

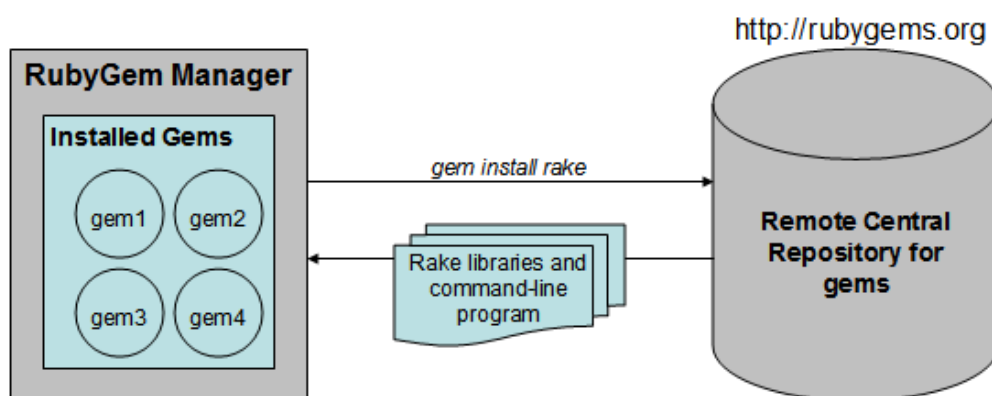
In this chapter, `<ruby_home>` refers to the location where Ruby is installed and `<app_home>` refers to the home directory of an application created with Rails.

### 7.2.1.1 RubyGems: Locating, installing and upgrading Ruby packages

**RubyGems** is the standard packaging and installation framework for Ruby libraries and applications. RubyGems makes it easy to locate, install, upgrade, and uninstall Ruby packages.

A gem, in the RubyGems world, is a packaged Ruby application or library. These files follow a standard format and are stored in a central repository on the Internet. Gems have a name and a version; for example *rake 0.4.16*.

A command line tool also called *gem* can be used to manipulate these gem files. If you want a gem, you can simply ask RubyGems to install it (and all its dependencies) online. Everything is done for you. *Figure.7.6* shows you how to install the "rake" gem through the internet.



**Figure 7.6 – Installing the rake gem**

As shown in the figure, from the machine where you installed Ruby, you can issue the command `gem install rake`. This will connect to <http://rubygems.org/>, and locate the **rake** libraries and the **rake** command-line program. We discuss the **rake** command later; for now, it's good to know that **rake** is a tool you can use to build other gems. Once the libraries and command-line program are downloaded, they are automatically installed.

## 7.2.2 Installing Rails

With RubyGems you can install *Rails* and its dependencies easily just as other gems. Install Rails by following these steps:

1. Open a Command Prompt window.

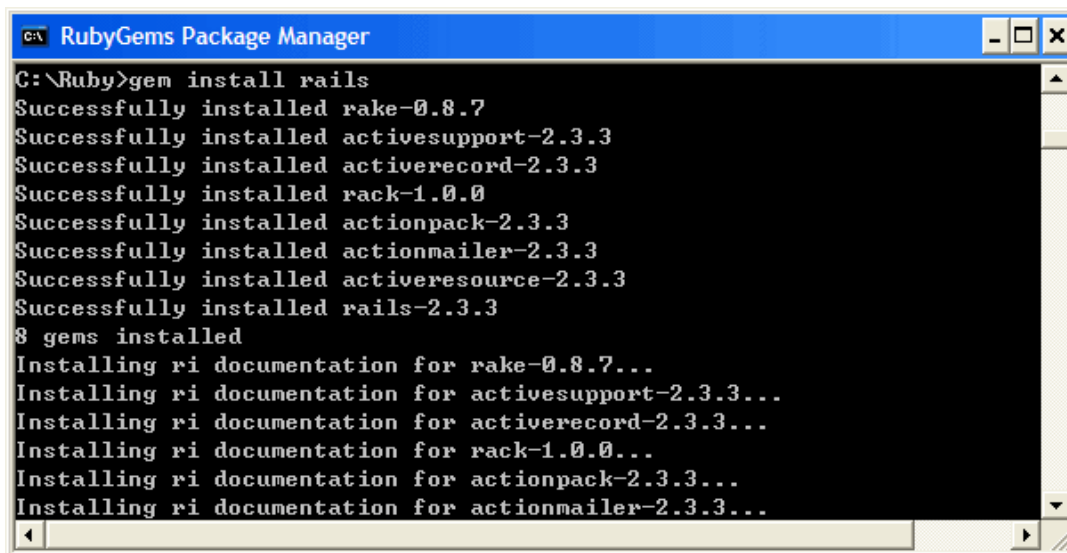
*Start -> All Programs -> Accessories -> Command Prompt*

2. Change the directory to **<ruby\_home>** and issue the gem command:

```
gem install rails
```

It may take several minutes to get all packages installed depending on your network speed. The gem command accesses the <http://rubygems.org/> site to look for the required Rails

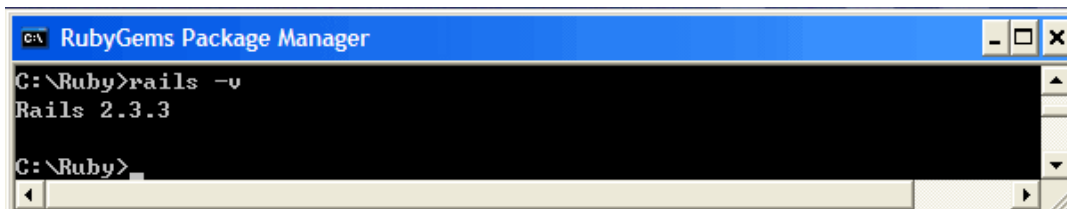
packages. Once downloaded and installed successfully, you'll see the result as shown in *Figure 7.7*



```
C:\Ruby>gem install rails
Successfully installed rake-0.8.7
Successfully installed activesupport-2.3.3
Successfully installed activerecord-2.3.3
Successfully installed rack-1.0.0
Successfully installed actionpack-2.3.3
Successfully installed actionmailer-2.3.3
Successfully installed activerecord-2.3.3
Successfully installed rails-2.3.3
8 gems installed
Installing ri documentation for rake-0.8.7...
Installing ri documentation for activesupport-2.3.3...
Installing ri documentation for activerecord-2.3.3...
Installing ri documentation for rack-1.0.0...
Installing ri documentation for actionpack-2.3.3...
Installing ri documentation for actionmailer-2.3.3...
```

**Figure 7.7** Installing Rails

To test whether you have successfully installed Ruby on Rails or not, issue the command `rails` with the `-v` flag. This command will report the Rails version as shown in *Figure 7.8*.



```
C:\Ruby>rails -v
Rails 2.3.3
C:\Ruby>
```

**Figure 7.8** – Rails Version

### 7.2.3 Creating your first RoR application and starting the Web server

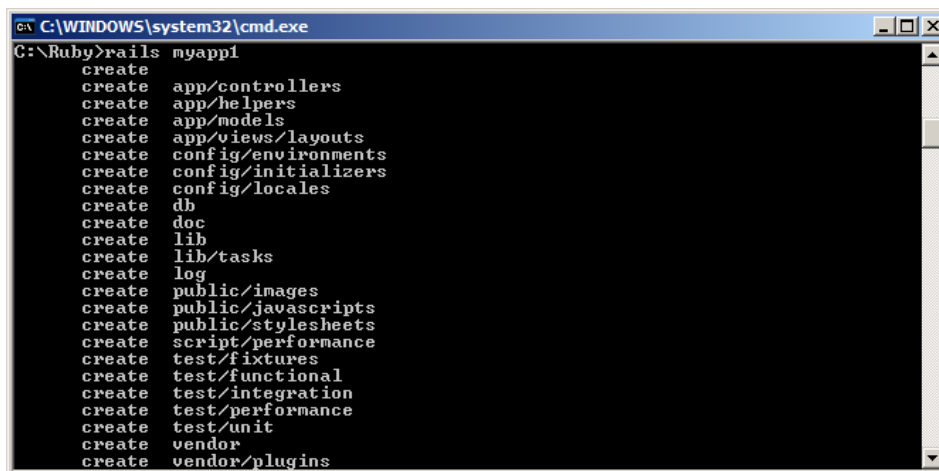
Let's create your very first RoR application. It will be so simple you don't even have to write any code! Here are the steps:

1. Change the directory to `<ruby_home>`: `cd <ruby_home>`
2. Create your RoR application named `myapp1` by issuing the following command

```
rails myapp1
```

This will create a structure for your application as shown in *Figure 7.9*. However, at this point we will not write any code.





```
C:\WINDOWS\system32\cmd.exe
C:\Ruby>rails myapp1
create
create  app/controllers
create  app/helpers
create  app/models
create  app/views/layouts
create  config/environments
create  config/initializers
create  config/locales
create  db
create  doc
create  lib
create  lib/tasks
create  log
create  public/images
create  public/javascripts
create  public/stylesheets
create  script/performance
create  test/fixtures
create  test/functional
create  test/integration
create  test/performance
create  test/unit
create  vendor
create  vendor/plugins
```

Figure 7.9 - Creating the RoR application myapp1

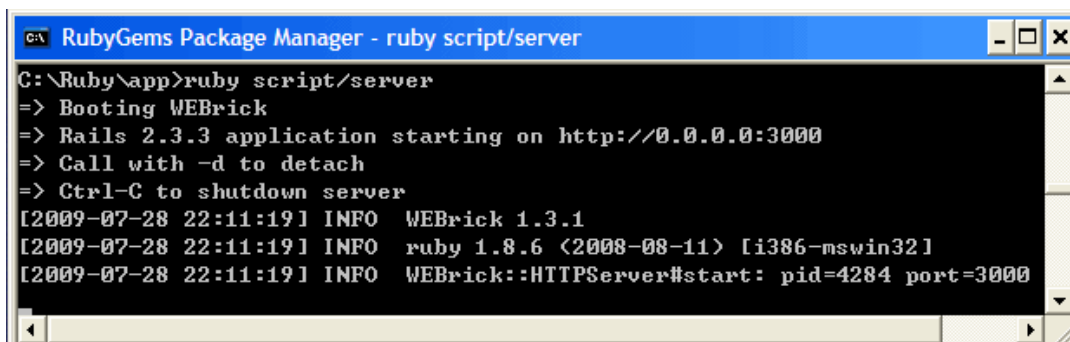
3. Change the work directory to `<app_home>`, the directory for your application:

```
cd myapp1
```

4. Start the WEBrick Web Server:

```
ruby script/server
```

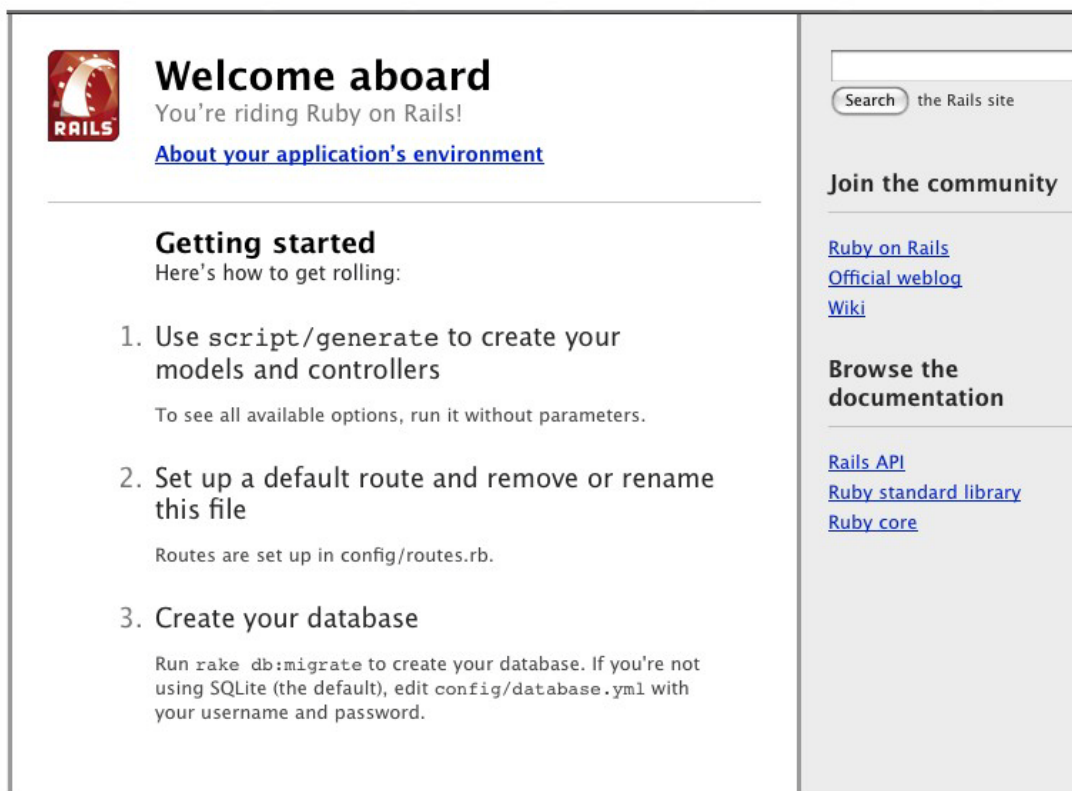
When the Web server starts, it will report the process ID (PID) and TCP/IP listening port number which should default to 3000. This is illustrated in *Figure 7.10*.



```
RubyGems Package Manager - ruby script/server
C:\Ruby\app>ruby script/server
=> Booting WEBrick
=> Rails 2.3.3 application starting on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2009-07-28 22:11:19] INFO  WEBrick 1.3.1
[2009-07-28 22:11:19] INFO  ruby 1.8.6 (2008-08-11) [i386-mswin32]
[2009-07-28 22:11:19] INFO  WEBrick::HTTPServer#start: pid=4284 port=3000
```

Figure 7.10 - Starting the WEBrick Web Server

Now that the Web server is up and running, open a browser and redirect it to <http://localhost:3000/> which will take you the Welcome page shown in *Figure 7.11*.



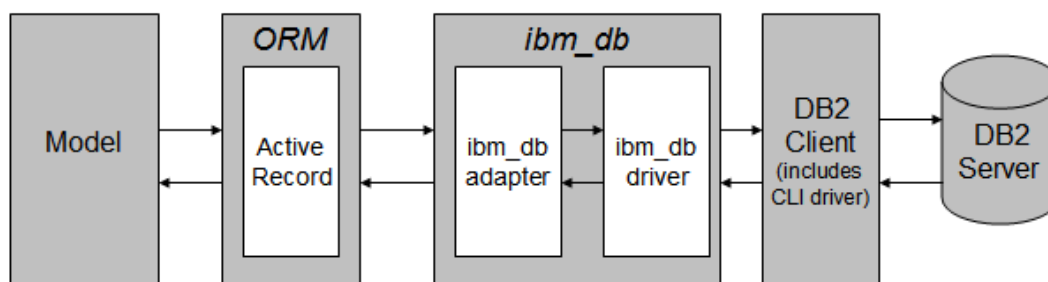
**Figure 7.11 - Welcome Page**

Thus far we have created the structure to develop a test application `myapp1`, and started the Web server. In a later section we show you how to work with a DB2 database and create models, views and controllers.

#### 7.2.4 Working with a DB2 database: The `ibm_db` gem

Rails support for DB2 is provided directly from IBM itself through an open source gem called `ibm_db`. This gem contains a driver (written in C) that allows Ruby to communicate with DB2, and an adapter written in Ruby that enables Active Record to work with DB2.

The `ibm_db` adapter has a direct dependency on the `ibm_db` driver, which utilizes the IBM Driver for ODBC and CLI to connect to IBM data servers. As mentioned earlier, Active Record is the object-relational mapping (ORM) layer supplied with Rails. It closely follows the standard ORM model but differs from most other ORM libraries in the way it is configured. By using a sensible set of defaults, Active Record minimizes the amount of configuration that developers need to perform. *Figure 7.12* illustrates how Active Record communicates with IBM\_DB and DB2 Server.



**Figure 7.12 – Interactions between Active Record and IBM\_DB**

With the `ibm_db` gem installed, Rails can be used with a DB2 data server, including DB2 Express-C and other DB2 editions on Linux, UNIX and Windows; DB2 for i5/OS®, and DB2 for z/OS.

#### 7.2.4.1 Installing the `ibm_db` gem

To install the `ibm_db` gem on Windows, simply run this command:

```
gem install ibm_db
```

The above command retrieves a pre-built binary version of the driver, so you won't need any compilers or build tools. If prompted for a version from a list, select the latest `mswin32` option available. *Figure 7.13* illustrates how to run this command from a Windows Command Prompt, and part of the output.

```

C:\Ruby>gem install ibm_db
Successfully installed ibm_db-1.1.0-x86-mswin32
1 gem installed
Installing ri documentation for ibm_db-1.1.0-x86-mswin32...
Installing RDoc documentation for ibm_db-1.1.0-x86-mswin32...
C:\Ruby>
  
```

**Figure 7.13 – Installing the `ibm_db` gem**

On Linux and UNIX, the gem is built from source; because of this, the installation process requires a few more steps as shown below:

1. Open up a shell as root by running:

```
$ sudo -s
```

2. Set the environment. The assumption is that `db2inst1` is the DB2 instance owner:

```
$ export IBM_DB_INCLUDE=/home/db2inst1/sqllib/include
```

```
$ export IBM_DB_LIB=/home/db2inst1/sqllib/lib
```

3. Source the DB2 profile. This step is technically not required to build the driver, but it enables a user other than the DB2 instance owner to execute commands like `db2` or `db2level`.

```
$ . /home/db2inst1/sqllib/db2profile
```

4. Finally, install `ibm_db` by executing from the same shell:

```
$ gem install ibm_db
```

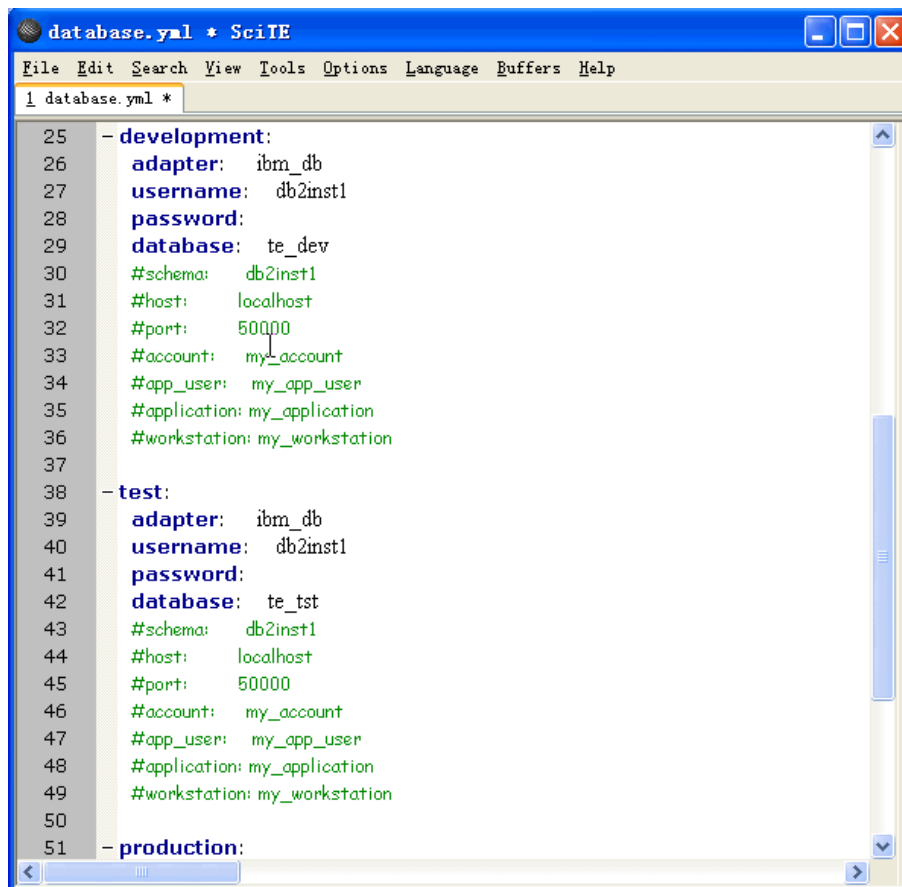
5. Once the gem is safely installed, you can exit from the root shell with `exit`.

On Mac OS X Snow Leopard (a Tiger version of DB2 doesn't exist), you can follow these steps:

1. `$ sudo -s`
2. `$ export IBM_DB_INCLUDE=/Users/myuser/sqllib/include`
3. `$ export IBM_DB_LIB=/Users/myuser/sqllib/lib64`
4. `$ export ARCHFLAGS="-arch x86_64"`
5. `$ gem update --system`
6. `$ gem install ibm_db`
7. `$ exit`

#### 7.2.4.2 The `database.yml` configuration file

After installing Ruby on Rails successfully and creating a new Rails application, the configuration file **`database.yml`** would be automatically created in the directory `C:\<ruby_home>\<app_home>\config`. Using the **`myapp1`** application created earlier, this file would be located under `C:\Ruby\myapp1\config`. The `database.yml` file provides information to your application about how to work with DB2. *Figure 7.14* illustrates a sample `database.yml` file opened with the SciTE editor that was installed with Ruby (*Start -> All Programs -> Ruby-186-27 -> SciTE*).



```

25 - development:
26   adapter:  ibm_db
27   username: db2inst1
28   password:
29   database: te_dev
30   #schema:  db2inst1
31   #host:    localhost
32   #port:    50000
33   #account: my_account
34   #app_user: my_app_user
35   #application: my_application
36   #workstation: my_workstation
37
38 - test:
39   adapter:  ibm_db
40   username: db2inst1
41   password:
42   database: te_tst
43   #schema:  db2inst1
44   #host:    localhost
45   #port:    50000
46   #account: my_account
47   #app_user: my_app_user
48   #application: my_application
49   #workstation: my_workstation
50
51 - production:

```

**Figure 7.14 – Contents of a sample database.yml configuration file**

As you can see from *Figure 7.14*, the `database.yml` file contains three different sections (development, test and production) in which Rails can run by default. The development environment is used to interact manually with the application; the test environment is used to run automated tests; the production environment is used when you deploy your application for the world to use.

*Table 7.2* provides a description of some of the attributes included in the `database.yml` file

| Connection attribute | Description                                          | Required |
|----------------------|------------------------------------------------------|----------|
| Adapter              | Ruby adapter name, for DB2 it is <code>ibm_db</code> | Yes      |

|             |                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Database    | Database alias/name of the database to which the Rails project connects                                                                                               | Yes                                                                                                                                                                                                                                                                                                                                                                                              |
| Username    | User ID used to connect to the DB2 database                                                                                                                           | Yes                                                                                                                                                                                                                                                                                                                                                                                              |
| Password    | Password for the user ID specified                                                                                                                                    | Yes                                                                                                                                                                                                                                                                                                                                                                                              |
| Schema      | The collection of named objects. The schema provides a way to group objects logically within the database.                                                            | Optional. If the schema is missing from database.yml, this attribute is set to the authorization ID of the current session user.                                                                                                                                                                                                                                                                 |
| Application | Application name. Only applicable when working with DB2 Connect™ software to work with DB2 for i5/OS and DB2 for z/OS.                                                | Optional                                                                                                                                                                                                                                                                                                                                                                                         |
| Account     | A character string used to identify the client accounting string. Only applicable when working with DB2 Connect software to work with DB2 for i5/OS and DB2 for z/OS. | Optional                                                                                                                                                                                                                                                                                                                                                                                         |
| Workstation | A character string used to identify the client workstation name. Only applicable when working with DB2 Connect software to work with DB2 for i5/OS and DB2 for z/OS.  | Optional                                                                                                                                                                                                                                                                                                                                                                                         |
| Host        | Host name of the remote server where the database resides                                                                                                             | The optional connection attributes <i>host</i> and <i>port</i> associated with remote TCP/IP connections are only required when DB2 catalog information is not available and no data source has been registered in the db2cli.ini configuration file for DB2 CLI. This type of setup is possible while using the IBM Driver for ODBC and CLI instead of a complete DB2 Client installed locally. |

|      |                                                                  |                                                 |
|------|------------------------------------------------------------------|-------------------------------------------------|
| Port | This parameter contains the name of the DB2 instance TCP/IP port | Same as for the Host attribute described above. |
|------|------------------------------------------------------------------|-------------------------------------------------|

**Table 7.2 - Description of some DB2 connection attributes in database.yml file**

## 7.3 Developing RoR applications

This section describes how to build simple Web applications using Rails **scaffolding**. Using the **scaffold** utility, you can quickly generate some of the major pieces of an application. The scaffold utility will easily create the models, views, and controllers for a new resource in a single operation.

The next sections describe the steps to create a simple Web application using scaffolding. The application will access a DB2 database; therefore, you first need to create a DB2 database, and configure the database.yml file.

### 7.3.1 Developing a sample application: A book catalog

The book catalog application, which for simplicity is named **bookapp**, allows users to obtain information about books. There will be two types of users: Regular users, and authors. The requirements of the **bookapp** application are simple:

1. Regular users can browse information about books.
2. Authors are regular users who can add new books to the catalog and update book information.
3. There will be one page to display books, and one page to manage books.
4. XML will be used to store book's information.

This application only needs one table, the **books** table. Its columns are described below, in *Table 7.3*.

| Column Name | Data Type | Description                                                       |
|-------------|-----------|-------------------------------------------------------------------|
| TITLE       | Varchar   | Book's name                                                       |
| DESCRIPTION | XML       | Book's information, including authors and a short description.    |
| STATUS      | Varchar   | Indicates whether a book is available, completed, not ready, etc. |

**Table 7.3 – books table columns and description**

The table will be created automatically in DB2 through RoR using the **scaffold** and **rake** commands as we will describe soon. RoR will also add extra columns, one of them being the ID column which would be used as the primary key.

The two pages mentioned in requirement #3 above will correspond to two resources: **book** and **catalog** which we will build later with the **scaffold** command.

In this particular example, the RoR application, the WebBrick server, and the DB2 server are all running on the same Windows machine. Therefore, when configuring the `database.yml` file, we will specify localhost.

### 7.3.1.1 Creating a DB2 database

Let's create the **booksdb** database using the following command from the DB2 Command Window, or Linux/UNIX shell:

```
db2 create db booksdb using codeset utf-8 territory us
```

This may take few minutes as DB2 is creating some internal objects, and performing some autoconfiguration. As mentioned earlier, you do not need to create the **books** table using DB2 tools. This will be created through RoR.

### 7.3.1.2 Creating the application structure

Let's create the **bookapp** application by opening a terminal, navigating to a folder where you have rights to create files (we refer to it as **<app\_home>** from now on), and typing the following:

```
rails -d ibm_db bookapp
```

With this command, the **bookapp** application is created. The **-d ibm\_db** indicates it should be preconfigured for a DB2 database using the **ibm\_db** adapter. *Figure 7.16* shows you the output of the command.



```

C:\Ruby>rails -d ibm_db bookapp
create
create   app/controllers
create   app/helpers
create   app/models
create   app/views/layouts
create   config/environments
create   config/initializers
create   config/locales
create   db
create   doc
create   lib
create   lib/tasks
create   log
create   public/images
create   public/javascripts
create   public/stylesheets
create   script/performance
create   test/fixtures
create   test/functional
create   test/integration
create   test/performance
create   test/unit
create   vendor
create   vendor/plugins
create   tmp/sessions
create   tmp/sockets
create   tmp/cache
create   tmp/pids
create   Rakefile
create   README
create   app/controllers/application_controller.rb
create   app/helpers/application_helper.rb
create   config/database.yml
create   config/routes.rb
create   config/locales/en.yml
create   db/seeds.rb
create   config/initializers/backtrace_silencers.rb
create   config/initializers/inflections.rb

```

**Figure 7.16 – Creating the Rails application *bookapp***

After creating the *bookapp* application, a folder with the name of the application is created in your working directory. Switch to that folder. In our example use this command:

```
cd <app_home>\bookapp
```

You will note several directories and files were automatically created under that directory. *Table 7.4* explains the main sub-directories that are created.

| Directory | Descriptions                                                                                                                            |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------|
| app\      | This directory contains the controllers, models, views, and helpers for your application.                                               |
| config\   | This directory includes files and subdirectories where you will configure your application's runtime rules, routes, database, and more. |
| db\       | Under this directory you will find files and subdirectories used to show                                                                |

|  |                                                                   |
|--|-------------------------------------------------------------------|
|  | your current database schema, as well as the database migrations. |
|--|-------------------------------------------------------------------|

**Table 7.4 - Default directory structure for a RoR application**

As explained earlier, you must configure the `database.yml` file to enter the values that will tell the RoR application how to connect to the DB2 database. Edit the `<app_home>\bookapp\config\database.yml` file as follows under the development section:

```
development:
  adapter:      ibm_db
  username:     arfchong
  password:     passwd
  database:     booksdb
  host:         localhost
  port:         50000
```

We use the `ibm_db` adapter, and want to connect to the `booksdb` database. In this example the database server resides on the same machine as the RoR application, therefore the host and port information could be removed, and RoR would establish a local connection to the DB2 database. For illustration purposes, we have set the `host` field to `localhost`, and the `port` field to `50000`. This means that RoR will establish the connection using TCPIP as if the DB2 server was on a different, remote machine.

The userID and password to use should be defined at the remote server. In our example, the username is `arfchong`, and the password is `passwd`. If you are following along, you'd have to change these values appropriately.

Since we are working on the development of this application, you can remove or comment out the other two sections in `database.yml`: `test`, and `production`.

To test you can connect to the database with this configuration, issue the following command:

```
<app_home>\bookapp> rake db:migrate
```

If you get an error, there is something wrong with the configuration values in `database.yml`, or maybe the DB2 instance is not running. An error means that Rails can't work with your database.

Next, we use scaffolding to quickly generate a template for our application. You can later modify this template according to your needs. Run the `scaffold` command as follows to create the `book` resource with a table that has the columns title, description and status.

```
ruby <app_home>\bookapp\script\generate scaffold Book
      title:string description:xml status:string
```

Figure 7.17 below illustrates the output of running the above command.

```

C:\Ruby\bookapp>ruby script\generate scaffold Book title:string description:xml
status:string
exists app/models/
exists app/controllers/
exists app/helpers/
create app/views/books
exists app/views/layouts/
exists test/functional/
exists test/unit/
create test/unit/helpers/
exists public/stylesheets/
create app/views/books/index.html.erb
create app/views/books/show.html.erb
create app/views/books/new.html.erb
create app/views/books/edit.html.erb
create app/views/layouts/books.html.erb
create public/stylesheets/scaffold.css
create app/controllers/books_controller.rb
create test/functional/books_controller_test.rb
create app/helpers/books_helper.rb
create test/unit/helpers/books_helper_test.rb
create route
map.resources :books
dependency model
exists app/models/
exists test/unit/
exists test/fixtures/
create app/models/book.rb
create test/unit/book_test.rb
create test/fixtures/books.yml
create db/migrate
create db/migrate/20100611202808_create_books.rb

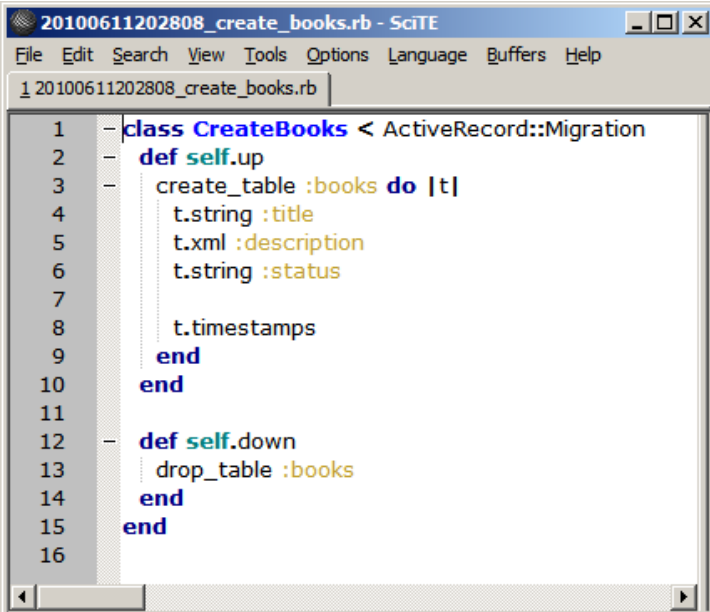
C:\Ruby\bookapp>

```

Figure 7.17 – Using scaffold to quickly create the *bookapp* sample application

Many components have been done by scaffold, saving us a lot of time. The main two that we are interested in are the model itself, `book.rb` and the migrate file, `20100611202808_create_books.rb` which are highlighted in *Figure 7.17* above.

Let's take a look at the migrate file `20100611202808_create_books.rb`. This generated file includes a timestamp as part of the file name. The `rb` extension identifies the file as a Ruby file. *Figure 7.18* below shows the contents of this file.



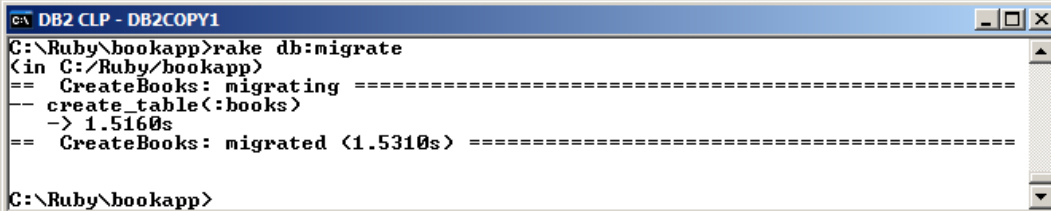
```
1 -- class CreateBooks < ActiveRecord::Migration
2 --   def self.up
3 --     create_table :books do |t|
4 --       t.string :title
5 --       t.xml :description
6 --       t.string :status
7 --
8 --       t.timestamps
9 --     end
10 --   end
11 --
12 --   def self.down
13 --     drop_table :books
14 --   end
15 -- end
16
```

Figure 7.18 – Contents of the migration file `20100611202808_create_books.rb`

The `up` method is used when applying the migration to DB2. This defines how the table will be created. The `down` method undoes what `up` has done. It is executed when you want to revert the database to a previous version.

Earlier we ran the `rake db:migrate` command to test if the `database.yml` file was configured correctly to connect to the database. What this command actually does is not only connect to the database, but also review the contents of migration files (if they exist), and apply the required changes. Now that we created a migration file `20100611202808_create_books.rb` with the `scaffold` utility, the `rake` command will apply all the migration instructions that have not been applied to the database before from this file. In this case, since the `books` table was not created before, it will create it in DB2.

The `db:migrate` parameter specified in the command indicates that the required commands and SQL statements should be generated and passed to DB2. *Figure 7.19* shows the output of this command.



```
C:\Ruby\bookapp>rake db:migrate
(in C:/Ruby/bookapp)
== CreateBooks: migrating =====
-- create_table(:books)
-> 1.5160s
== CreateBooks: migrated (1.5310s) =====

C:\Ruby\bookapp>
```

Figure 7.19 – Running `rake db:migrate` command

Let's verify in DB2 that the table was created using the DB2 command `list tables` from the DB2 Command Window or Linux/UNIX shell. *Figure 7.20* shows the command used and the output.

```

C:\Ruby\bookapp\db\migrate>db2 connect to booksdb

Database Connection Information
Database server      = DB2/NT 9.7.2
SQL authorization ID = ARFCHONG
Local database alias = BOOKSDB

C:\Ruby\bookapp\db\migrate>db2 list tables
Table/View          Schema           Type  Creation time
-----
BOOKS               ARFCHONG        I     2010-06-11-16.34.13.437002
SCHEMA_MIGRATIONS  ARFCHONG        I     2010-06-11-16.27.23.031001

  2 record(s) selected.

C:\Ruby\bookapp\db\migrate>_

```

**Figure 7.20 – Listing from DB2 the tables created by rake db:migrate**

*Figure 7.20* shows that `rake db:migrate` command created the `BOOKS` table, but also a table called `SCHEMA_MIGRATIONS`. This second table is used by RoR to track migration instructions for your database.

Let's verify the structure of the `BOOKS` table using the DB2 command `describe table` as shown in *Figure 7.21*.

```

C:\Ruby\bookapp>db2 describe table books
Column name          Data type schema  Data type name  Column Length  Scale  Nulls
-----
ID                   SYSIBM         INTEGER         4              0     No
TITLE                SYSIBM         VARCHAR         255            0     Yes
DESCRIPTION          SYSIBM         XML             0              0     Yes
STATUS               SYSIBM         VARCHAR         255            0     Yes
CREATED_AT           SYSIBM         TIMESTAMP      10             6     Yes
UPDATED_AT           SYSIBM         TIMESTAMP      10             6     Yes

  6 record(s) selected.

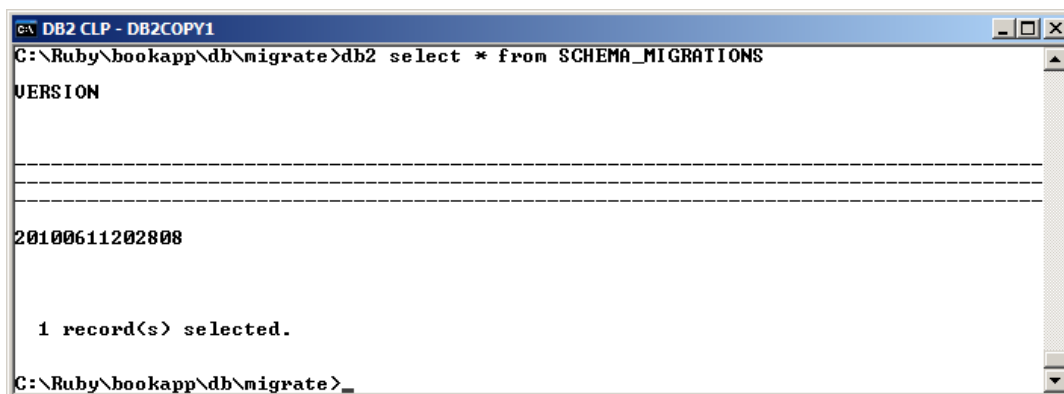
C:\Ruby\bookapp>_

```

**Figure 7.21 – The structure of the BOOKS table created by rake db:migrate**

Note that in *Figure 7.21* there are three new columns: `ID`, `CREATED_AT`, `UPDATED_AT`. Rails automatically creates a default primary key column `ID`; therefore, you do not have to explicitly define a primary key when working with Rails. With respect to columns `CREATED_AT` and `UPDATED_AT`, Rails adds these columns because they are useful in many scenarios, but they are not necessarily needed for migrations. All the migration information that Rails needs is stored in the migration table.

Now let's take a look at the other table that was created, the SCHEMA\_MIGRATION table. *Figure 7.22* shows the contents of this table.



```
DB2 CLP - DB2COPY1
C:\Ruby\bookapp\db\migrate>db2 select * from SCHEMA_MIGRATIONS
VERSION
-----
20100611202808

1 record(s) selected.
C:\Ruby\bookapp\db\migrate>
```

**Figure 7.22 –Contents of the table SCHEMA\_MIGRATION**

As you can see from the figure, the SCHEMA\_MIGRATION table contains only one column, the VERSION column, which is used to keep track of migration version numbers. The version number is the numerical prefix on the migration's filename. For example, for file `20100611202808_create_books.rb`, the version is `20100611202808`.

The `rake db:migrate` in its simplest form will run the `up` method for all the migrations that have not yet been run. If you want a specific version, Active Record will run the required migrations (up or down) until it has reached the specified version. For example to migrate to version `20100705000000` run:

```
rake db:migrate VERSION=20100705000000
```

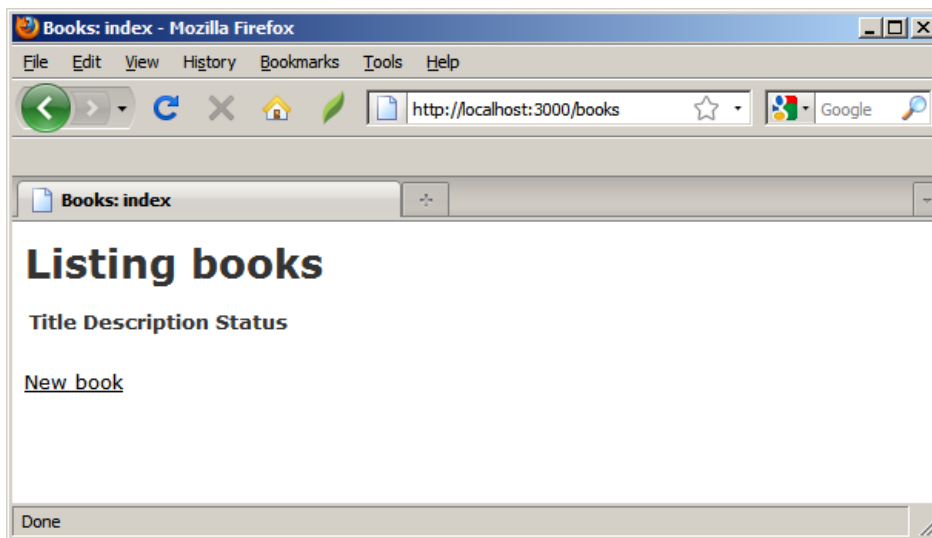
If this is greater than the current version, that is, it is migrating upwards; the `up` method will be invoked on all migrations up to and including `20100705000000`, if it is migrating downwards, the `down` method will be invoked on all the migrations down to, but not including, `20100705000000`. To rollback to the last migration, use:

```
rake db:rollback.
```

Let's take a look now at what has been built for this application. If the WEBrick Web server is not started, use this command to start it:

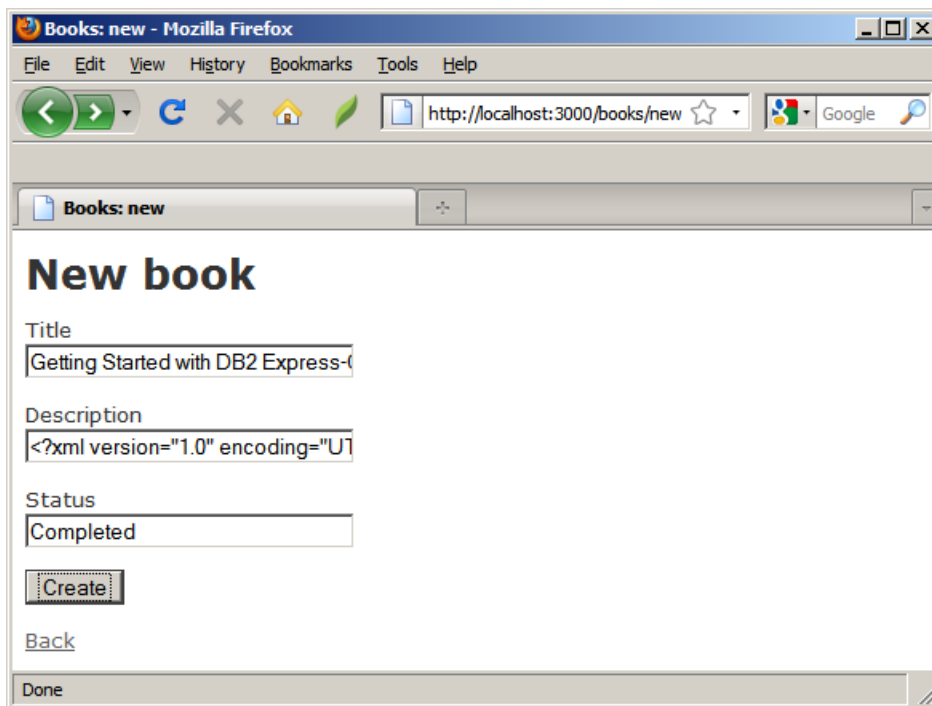
```
<app_home>\bookapp> ruby script/server
```

Then enter <http://localhost:3000/books> in a browser. *Figure 7.23* displays the output.



**Figure 7.23 – The Listing books page**

Click the new book link, and create an entry for a new book as shown in *Figure 7.24* below.



**Figure 7.24 – Creating a new book item**

Note that through the RoR application, the XML document is correctly stored in the BOOKS table as shown in *Figure 7.25* which uses IBM Data Studio.

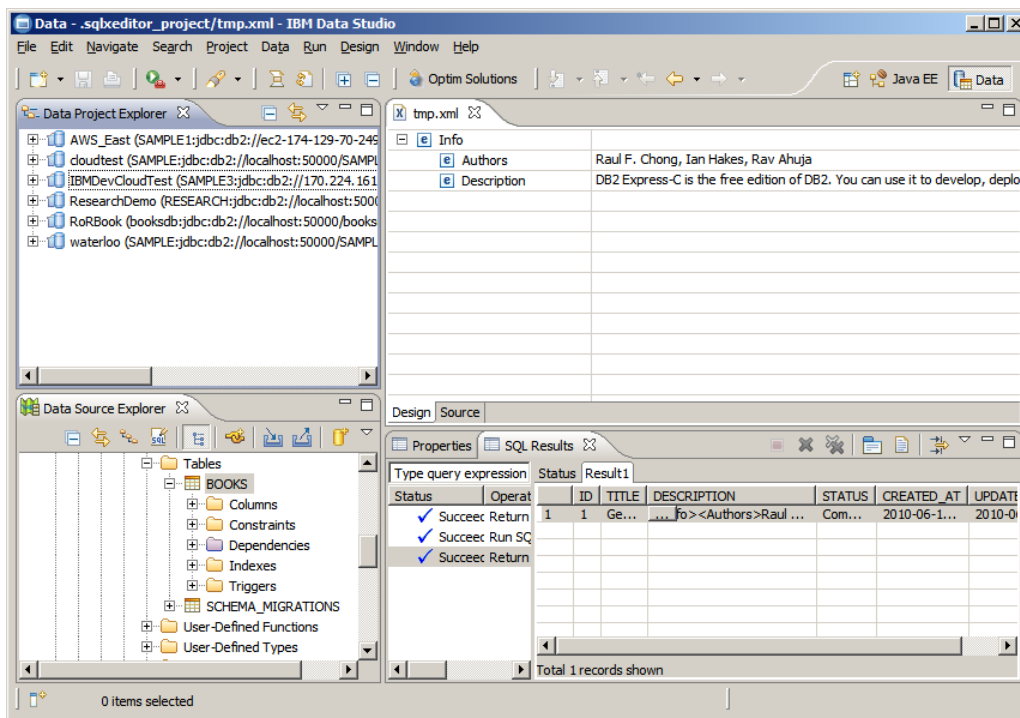


Figure 7.25 – Viewing the XML document from IBM Data Studio

If you point your browser back to <http://localhost:3000/books> you will now be able to see the book that was created. This is illustrated in Figure 7.26. You can also see other operations have been automatically created by RoR such as *Show*, *Edit* and *Destroy*.

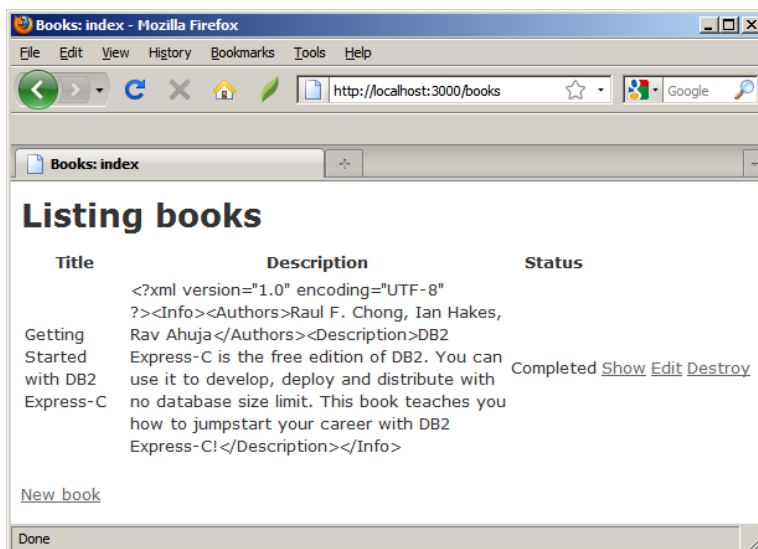


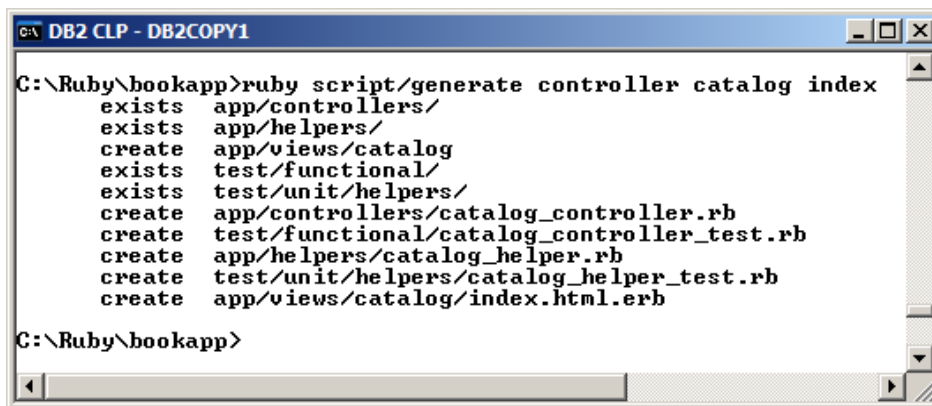
Figure 7.26 – Listing books



Now that we can add, delete and update books using the *book* resource, we need to work on another resource that allows you to list all books. Let's create the *catalog* resource for this purpose using this command:

```
<app_home>\bookapp>ruby script/generate controller catalog index
```

In the above command the controller's name is *catalog* and a default view page named *index* is also created. *Figure 7.27* shows the output of this command.



```

C:\Ruby\bookapp>ruby script/generate controller catalog index
exists  app/controllers/
exists  app/helpers/
create  app/views/catalog
exists  test/functional/
exists  test/unit/helpers/
create  app/controllers/catalog_controller.rb
create  test/functional/catalog_controller_test.rb
create  app/helpers/catalog_helper.rb
create  test/unit/helpers/catalog_helper_test.rb
create  app/views/catalog/index.html.erb

C:\Ruby\bookapp>

```

**Figure 7.27 – Generating the catalog resource**

**Note:**

Once the *catalog* resource has been created, you must restart the Web server, otherwise you may get errors related to the routers.

The following actions will build basic logic for the *catalog* controller to list all the books. Copy and paste the code shown in *Listing 7.1* below to the file `<app_home>\bookapp\app\models\book.rb`

```

(1) class Book < ActiveRecord::Base
(2)   def self.find_books
(3)     xquery=
           "select i.TITLE, i.ID, t.AUTHOR, t.INFO, i.STATUS from books i,
           xmltable('$DESCRIPTION/Info'
                   columns AUTHOR varchar(100) path 'Authors',
                           INFO varchar(400) path 'Description'
                   ) as t"
(4)     find_by_sql(xquery)
       end
     end

```

**Listing 7.1 - Contents of the book.rb file to list all the books**

In *Listing 7.1*:

- (1) Defines the Book class
- (2) The `find_books` method that we will use in the catalog.
- (3) The `xquery` statement to use in the `find_books` method. This `xquery` statement retrieves columns from two tables, the BOOKS table (using the alias of *i*); and the table created with the XMLTABLE function (using the alias of *t*). For more information about the XMLTABLE function, refer to *Chapter 2, DB2 pureXML*
- (4) A built\_in method `find_by_sql` that can be used to execute raw sql.

**Note:**

For more information about the Ruby on Rails API, review <http://api.rubyonrails.org/>.

Copy and paste the code shown in *Listing 7.2* below to the file

```
<app_home>\bookapp\app\controllers\catalog_controller.rb.
```

This tells the `catalog` controller where to fetch books.

```
(1) class CatalogController < ApplicationController
(2)   def index
      @books=Book.find_books
    end
  end
End
```

**Listing 7.2 - Contents of the catalog\_controller.rb file**

In *Listing 7.2*:

- (1) Defines the CatalogController class
- (2) An index method is defined. It gets an instance variable `@books` returned by `find_books`, which was defined in the model of Book.

Copy and paste the code shown in *Listing 7.3* to the file

```
<app_home>\bookapp\app\views\catalog\index.html.erb

<h1>DB2 on Campus Book Series</h1>
<% for book in @books -%>
<div class="entry" >
<h3><%= link_to h(book.title) ,book ,:id => book%></h3>
<h3>Authors</h3>
<%= book.author %>
<h3>Description</h3>
<%= book.info %>
```

```
<h3>Status</h3>
<span class="status" ><%= book.status %></span>
</div>
<% end %>
```

#### Listing 7.3 - Contents of the index.html.erb file

The above html code tells a view how to display a book's information.

Let's now take a look at our code in action! Point your browser to <http://localhost:3000/catalog>. It should display a page with the catalog list as illustrated in Figure 7.28 below.

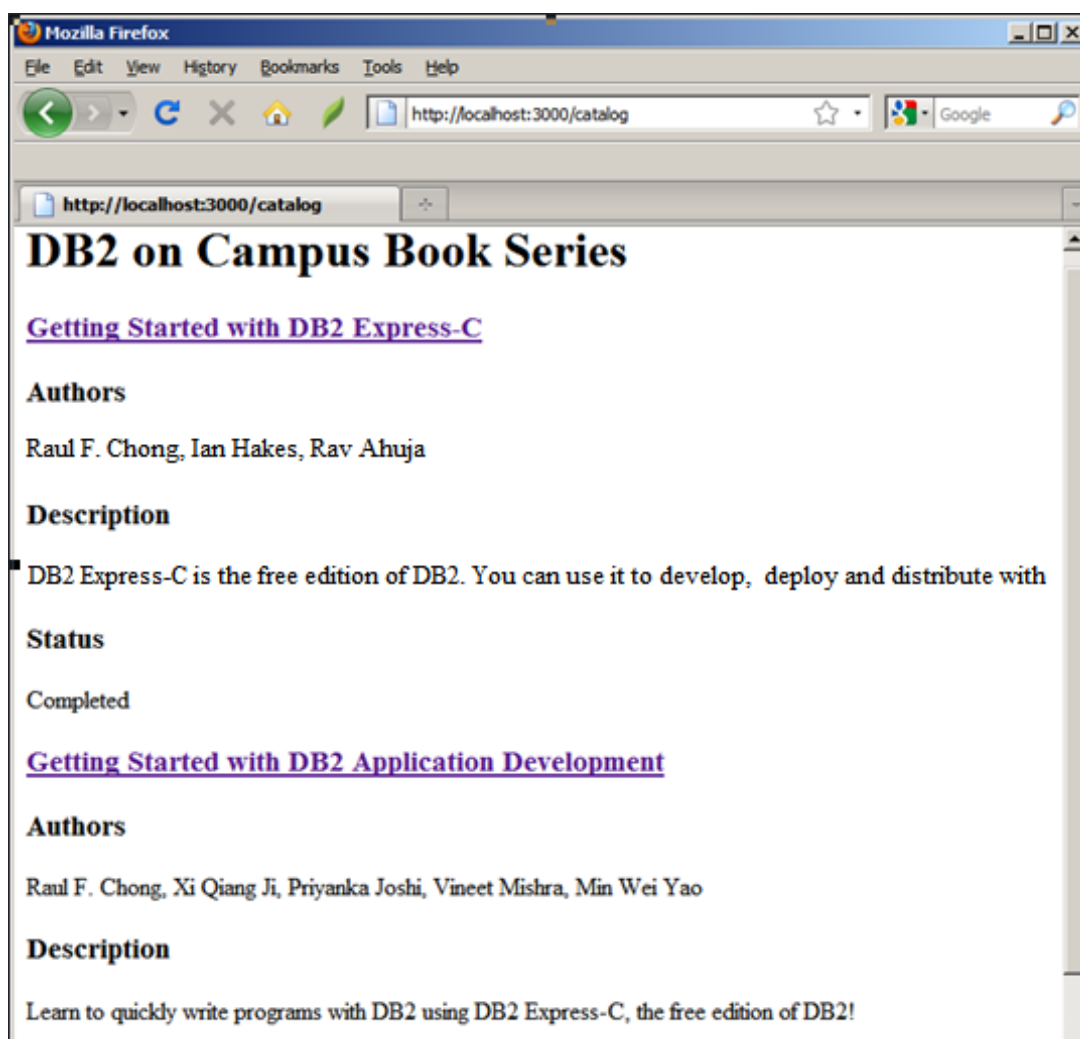


Figure 7.28 - Catalog list

### 7.3.2 Customizing the layout

Let's improve the appearance of the page using cascading style sheets (CSS). Browse to the directory `<app_home>\bookapp\public\stylesheets` and create a text file called `style.css`. Copy and paste the code shown in *Listing 7.4* below to the file `style`,

```
/* Styles for RoR - DB2 example */
#main {
margin-left: 10em;
padding-top: 1em;
padding-left: 2em;
background: white;
font-size: 14px;
font-family:Arial, sans-serif;
}

#side {
float: left;
padding-top: 1em;
padding-left: 1em;
padding-bottom: 1em;
width: 12em;
background: #152 ;
font-size: 12px;
font-family:Verdana, Arial, sans-serif;
}

#side a {
color: #bfb ;
font-size: small;
}

#banner {
background: #99cc66 ;
padding-top: 10px;
padding-bottom: 10px;
border-bottom: 1px solid;
font-size: 35px;
font-family:Verdana, Arial, sans-serif;
color: #282 ;
text-align: center;
}

#columns {
```

```
background: #152 ;
}

h1 {
font: 150% arial;
color: #230 ;
border-bottom: 3px solid #230 ;
}

```

#### Listing 7.4 - Contents of the file style - a CSS file

We will use this `style` CSS file when we change the *layout*. A layout in Rails is a template into which we can flow additional content. In our case, we can define a single layout for all the book pages and insert the catalog pages into that layout. There are many ways of specifying and using layouts in Rails. We only show you how to use the simplest method:

Create a template file in the `<app_home>\bookapp\app\views\layouts` directory with the same name as the controller `catalog`. Because layout is part of views, a suffix of `html.erb` must be added to `catalog` as in `catalog.html.erb`. All views rendered by the `catalog` controller will use this layout by default. Open a text editor and copy the code shown in *Listing 7.5* below and paste it into the `catalog.html.erb` file. This will create a new layout for all the pages under views of `catalog`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
  <title>DB Books Series</title>
(1) <%= stylesheet_link_tag "style"%>
</head>
<body id="catalog">
  <div id="banner">
    <%= @page_title || "DB2 on Campus Books Series"%>
  </div>
  <div id="side">
    <a href="http://localhost:3000/catalog" >Home</a><br />
(2)   <a href="http://localhost:3000/books" >Books</a><br />
    <br /><br /><br /><br /><br /><br /><br /><br /><br />
    <br /><br /><br /><br /><br />
  </div>
  <div id="main">
    <%= yield :layout %>
  </div>
</body>

```

```
</html>
```

**Listing 7.5 - Contents of the catalog.html.erb file**

In (1) note how we invoke the `style` CSS file.

In (2) we add a link to books, where all books information are manipulated, including create, update and delete.

Next, we will do the same for

```
<app_home>\bookapp\app\views\layouts\book.html.erb
```

Copy and paste the contents of *Listing 7.6* below which are very similar to *Listing 7.5*. Note that in *Listing 7.5* we had:

```
<a href="http://localhost:3000/books" >Books</a>
```

while in *Listing 7.6* we have:

```
<a href="http://localhost:3000/catalog" >Catalog</a>
```

This is done so each page points to each other in the Menu left bar.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <title>DB Books Series</title>
    <%= stylesheet_link_tag "style"%>
  </head>
  <body id="catalog">
    <div id="banner">
      <%= @page_title || "Books' Administration"%>
    </div>
    <div id="side">
      <a href="http://localhost:3000/catalog" >Home</a><br />
      <a href="http://localhost:3000/catalog" >Catalog</a>
      <br />
      <br /><br /><br /><br /><br /><br /><br /><br /><br />
      <br /><br /><br /><br /><br />
    </div>
    <div id="main">
      <%= yield :layout %>
    </div>
  </body>
</html>
```

**Listing 7.6 - Contents of the books.html.erb file**

We have now constructed a layout for all the pages. If you wanted to change this layout, you can simply change it in one place. This is a clear example of the DRY (Don't repeat yourself) RoR philosophy!

If you refresh the browser, you will see the improved look of the pages as illustrated in *Figure 7.28* to *Figure 7.30*

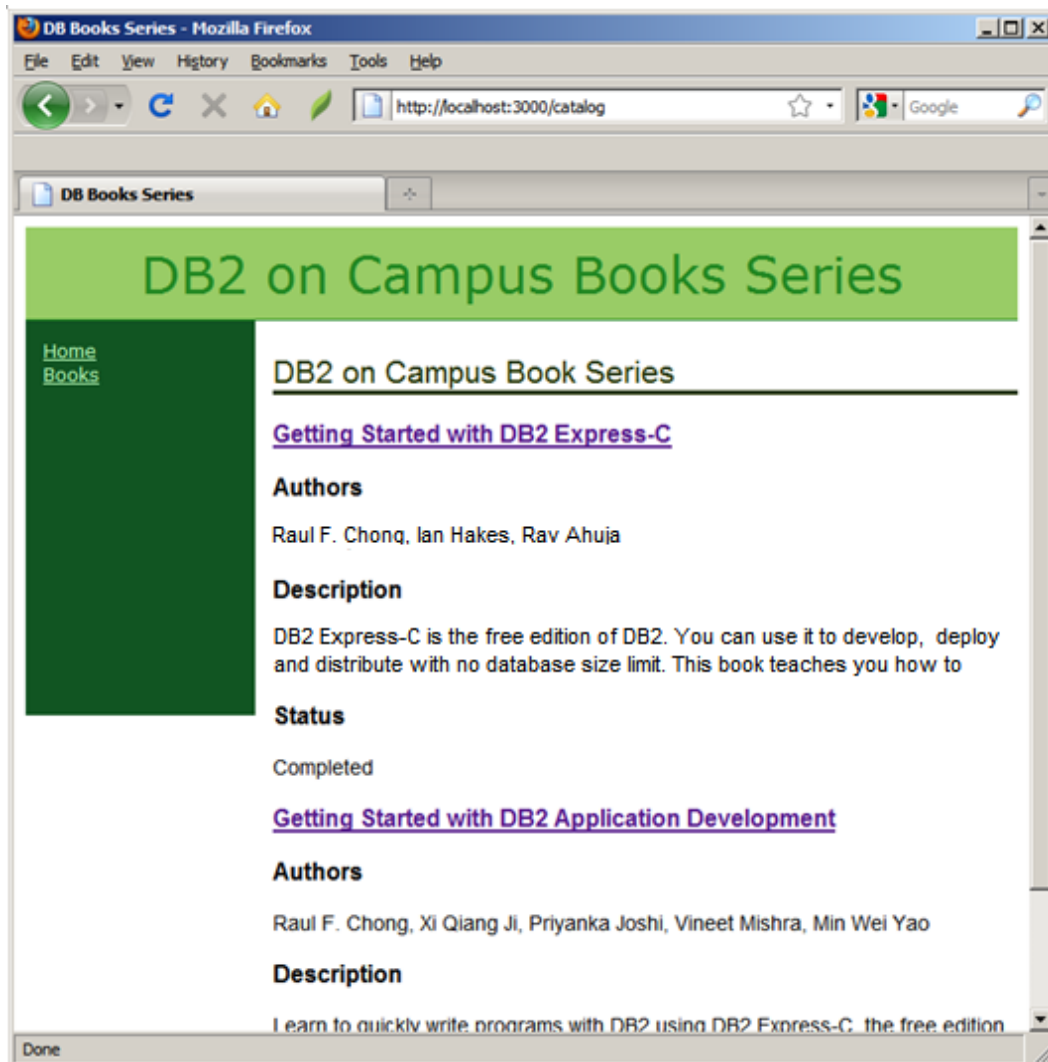


Figure 7.28 - The Catalog list after applying a layout

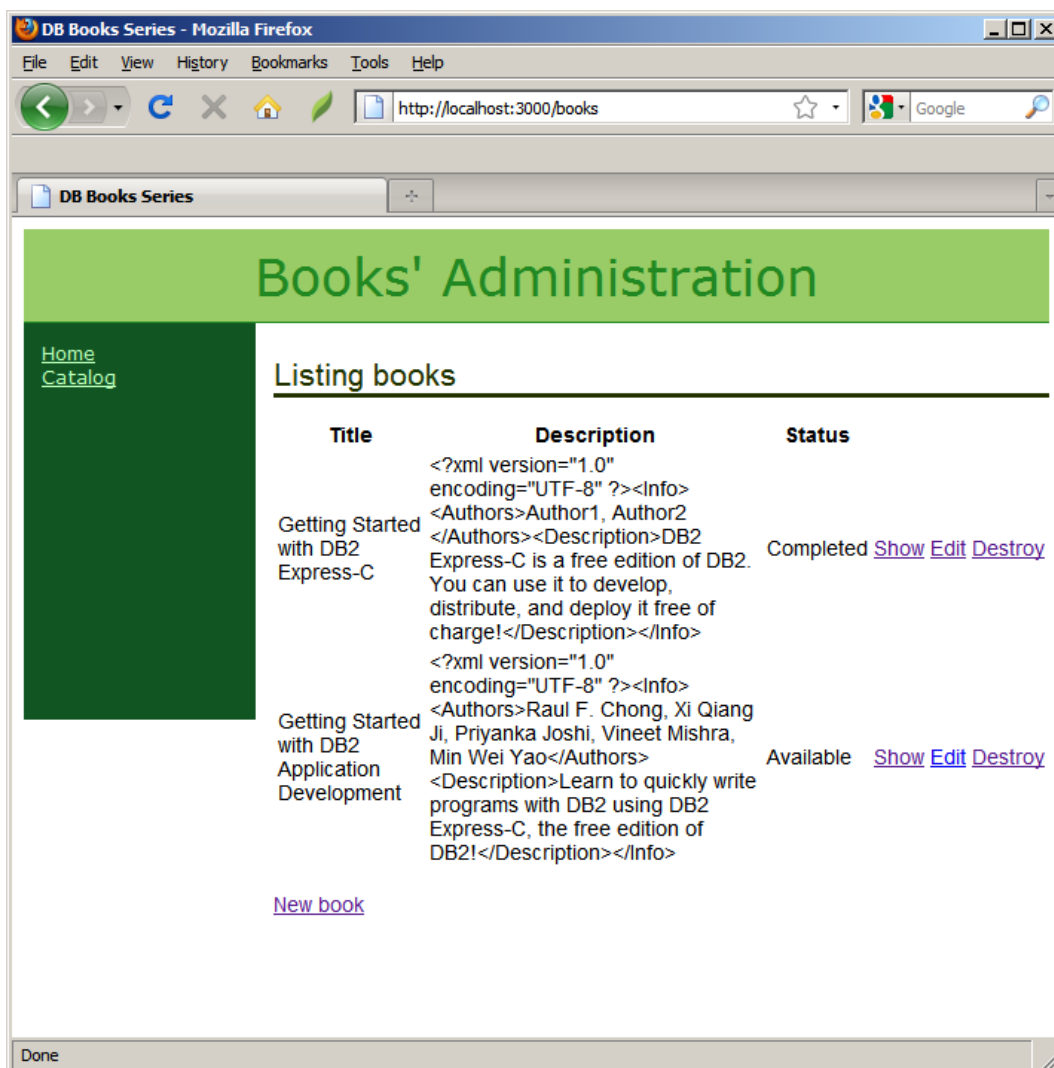


Figure 7.29 - The Book Administration page after applying a layout



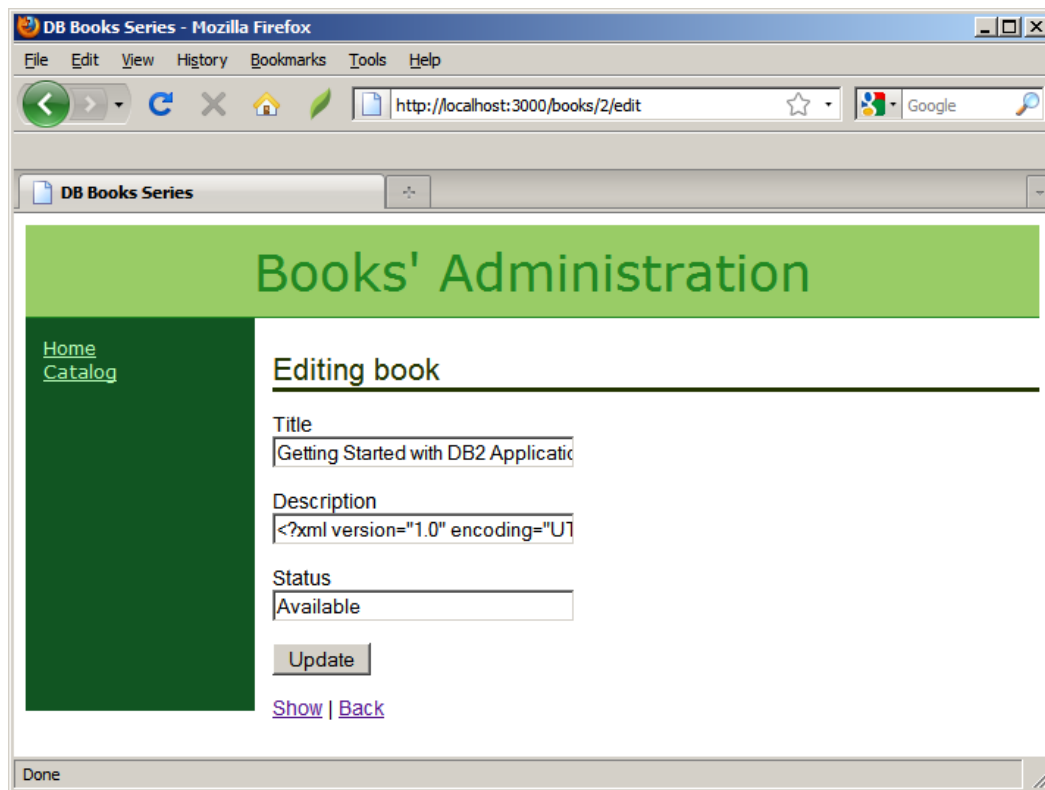


Figure 7.30 - The page to update books after applying a layout

## 7.4 Exercises

In this exercise, you will practice creating a blog application. Before running the exercise, make sure Ruby on Rails, and DB2 Express-C have been installed successfully. A table named POS, with three columns NAME, TITLE, CONTENT will be created during this process.

### Procedure:

1. Create the RoR blog application:

```
rails -d ibm_db blog
```

2. Configuring `ibm_db` according to your environment, such as local or remote

3. Create a resource

```
script/generate scaffold Post name:string title:string content:text
```

4. Running a migration

```
rake db:migrate
```

5. Start the Web server and click this link <http://localhost/posts>. Compare the result with files in the `Exercise_Files_DB2_Application_Development.zip` file that accompanies this book.

## 7.5 Summary

In this chapter, you have learned the basics to develop a Ruby on Rails application using the `ibm_db` adapter/driver and a DB2 data server. The chapter explained the setup required to get Ruby on Rails working on Windows, followed by the configuration of the `database.yml` file to connect to a DB2 database. At the end, the chapter showed how to develop a simple Web application. The sample application uses an XML column to store the description of each book.

## 7.6 Review questions

1. What is Ruby on Rails? What is a gem?
2. How can I install DB2's RoR driver and adapter?
3. How does RoR communicate with DB2?
4. How can I configure the `database.yml` file with DB2?
5. Can I use XML as a data type in RoR?
6. Which of the following statements about RubyGems is false?
  - A. A central repository for hosting packages in a standardized package format
  - B. Installs and manages multiple, and simultaneously installed versions of the same library.
  - C. The older versions of the same library must be deleted before the new one is installed
  - D. RubyGems are end-user tools for querying, installing, uninstalling, and manipulating packages.
  - E. None of the above
7. All the below items are gems but \_\_\_\_\_ is not.
  - A. Ruby
  - B. Rails
  - C. Rake
  - D. IBM\_DB
  - E. None of the above

8. Which of the following sentences about IBM\_DB and Active Record is true?
- A. IBM\_DB contains a driver and an adapter. The driver directly interacts with Active Record
  - B. Active Record is a part of IBM\_DB
  - C. IBM\_DB adapter utilizes the IBM Driver for **ODBC** and **CLI** to connect to IBM data servers.
  - D. To enable IBM\_DB, we should install both DB2 client and server on your PC.
  - E. None of the above
9. Which is optional in `database.yml` when we want to connect to a remote DB2 server installed on Linux?
- A. Account
  - B. Database
  - C. Password
  - D. Username
  - E. None of the above
10. Which of the following is the best sequence to install `ibm_db` on Linux?
- a. Open a shell as root
  - b. Set the environment
  - c. Install `ibm_db`
  - d. Source DB2 profile
  - e. Exit from the root shell
- A. a->b->c->d->e
  - B. a->b->d->c->e
  - C. a->d->b->c->e
  - D. a->c->d->b->e
  - E. None of the above



# 8

## Chapter 8 – Application development with PHP

*PHP (PHP Hypertext Preprocessor)* is an interpreted, general-purpose, open source, scripting programming language primarily intended for the development of Web applications. The first version of PHP was created by Rasmus Lerdorf and contributed under an open source license in 1995.

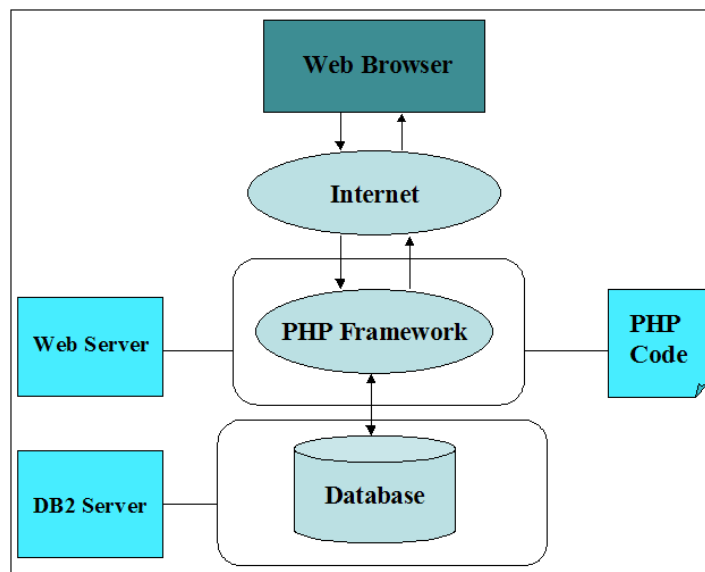
PHP generally runs on a Web server; its code is executed by the PHP runtime to create dynamic Web page content. It can also be used for command-line scripting and client-side GUI applications. PHP can be deployed on most Web servers, many operating systems and platforms, and can be used with many relational database management systems. It is available free of charge, and the PHP Group provides the complete source code for users to build, customize and extend for their own use.

In this chapter you will learn about:

- Setting up the PHP environment to work with DB2
- Developing PHP applications with DB2

### 8.1 PHP - DB2 Applications: The big picture

*Figure 8.1* provides an overview of a PHP-DB2 application. In the figure, a user opens a Web browser and points to a page that invokes a PHP script. The request is received by the Web server where PHP code also resides, and the PHP code is executed. If the request needs to access data from DB2, PHP will connect to the DB2 database and retrieve, update or delete the necessary data. After that, if output needs to be returned, it is sent as HTML to the Web browser.



**Figure 8.1 - Interaction between PHP, a Web server and DB2**

PHP primarily acts as a filter, taking input from a file or stream containing text and/or PHP instructions and outputs another stream of data; most commonly the output will be HTML. PHP scripts are often compiled at runtime by the PHP engine, which increases their execution speed. PHP scripts can also be compiled before runtime using PHP compilers.

## 8.2 Setting up the environment

To start working with PHP you need to install a Web server, PHP, and a DB2 database server. If your Web server and database server are on different machines, you also need to install a DB2 client on the Web server.

In this section we explain how to configure your environment manually by downloading and installing each component separately.

### 8.2.1 Setting up the PHP environment manually

If you would like to set up your environment manually by installing each component separately, you need to follow these steps:

1. Install the Apache HTTP Server

Download the source code from <http://httpd.apache.org/download.cgi> specific to your operating system. Installation instructions are available on the same URL.

2. Install DB2 Express-C. Download it from <http://www.ibm.com/db2/express>

3. On Linux, ensure you have the gcc compiler and the following development packages: apache-devel, autoconf, automake, bison, flex, gcc, and libxml2-devel package
4. If the Web server where the PHP code is and the database server are on different machines, download and install one of the following DB2 clients: IBM Data Server Driver Package, IBM Data Server Client, or IBM Data Server Driver for ODBC and CLI. These can be found at this location: <http://www-01.ibm.com/support/docview.wss?rs=4020&uid=swg21385217>
5. Download the latest version of the PHP from <http://www.php.net> for Linux/UNIX or Windows. The latest version of PHP at the time of writing is PHP 5.3. Install it, and configure it. The following sections explain how to do this on Linux/UNIX and Windows.

#### 8.2.1.1 Setting up the PHP environment manually on Linux or UNIX

1. Untar the PHP package previously downloaded by issuing the following command:  

```
tar -xjf php-5.x.x.tar.bz2
```
2. Change directories into the newly created `php-5.x.x` directory.
3. Configure the `makefile` by issuing the `configure` command. Specify the features and extensions you want to include in your custom version of PHP. A typical `configure` command includes the following options:

```
./configure --enable-cli--disable-cgi --with-apxs2=/usr/sbin/apxs2 --with-zlib --with-pdo-ibm=<sqllib>
```

Where `--with-pdo-ibm=<sqllib>` enables the `pdo_ibm` driver using the DB2 CLI library to access DB2 databases. The `<sqllib>` setting refers to the directory where DB2 was installed. The `pdo_ibm` driver is discussed in more detail later in this chapter.

4. Compile the files by issuing the `make` command.
5. Install the files by issuing the `make install` command. Depending on how you configured the PHP install directory using the `configure` command; you may need `root` authority to successfully issue this command. This should install the executable files and update the Apache HTTP Server configuration to support PHP.
6. Install the `ibm_db2` extension by issuing the following command as a user with `root` authority: `pecl install ibm_db2`

This command downloads, configures, compiles, and installs the `ibm_db2` extension for PHP.

7. Copy the `php.ini-recommended` file to the configuration file path for your new PHP installation. To determine the configuration file path, issue the `php -i` command and look for the `php.ini` keyword. Rename the file to `php.ini`.
8. Open the new `php.ini` file with a text editor and add the following lines, where "instance" refers to the name of the DB2 instance on Linux or UNIX.
  - To set the DB2 environment for `pdo_ibm`:

```
PDO_IBM.db2_instance_name=instance
```
  - To enable the `ibm_db2` extension and set the DB2 environment:

```
extension=ibm_db2.so
ibm_db2.instance_name=instance
```
9. Restart the Apache HTTP Server to enable the changed configuration.

#### 8.2.1.2 Setting up the PHP environment manually on Windows

1. Extract the PHP zip package previously downloaded into an install directory.
2. Extract the collection of **PECL** modules zip package into the `\ext\` subdirectory of your PHP installation directory.
3. Create a new file named `php.ini` in your installation directory by making a copy of the `php.ini-recommended` file.
4. Open the `php.ini` file in a text editor and add the following lines.
  - To enable the PDO extension and `pdo_ibm` driver:

```
extension=php_pdo.dll
extension=php_pdo_ibm.dll
```
  - To enable the `ibm_db2` extension:

```
extension=php_ibm_db2.dll
```
5. Enable PHP support in Apache HTTP Server 2.x by adding the following lines to your `httpd.conf` file, in which `phpdir` refers to the PHP install directory:

```
LoadModule php5_module 'phpdir/php5apache2.dll'
AddType application/x-httpd-php .php
PHPIniDir 'phpdir'
```
6. Restart the Apache HTTP Server to enable the changed configuration.



**Note:**

For Windows, it is always recommended to download the \*.msi installers instead of the zip archives for installation of both the Web server (Apache in our case) and PHP.

## 8.3 PHP - DB2 application development

In this section, we describe how to access a DB2 database from a PHP application. We first describe the different extensions that can be used for this purpose.

### 8.3.1 PHP extensions to use with DB2

There are three PHP extensions you can use with DB2 software ("DB2"), though only the first two are recommended ones:

- **ibm\_db2**

A PHP extension written, maintained, and supported by IBM for access to DB2 databases.

- **pdo\_ibm / pdo\_odbc**

A driver for the PHP Data Objects (PDO) extension that offers access to DB2 databases through the standard object-oriented database interface introduced in PHP 5.1. It uses CLI as the underlying interface to DB2. Alternatively, you can use pdo\_odbc which uses ODBC drivers or DB2 CLI.

- **Unified ODBC**

An extension that historically provided access to DB2 database systems. It is not recommended that you write new applications with this extension because **ibm\_db2** and **pdo\_ibm** both offer significant performance and stability benefits over Unified ODBC. We will not cover this extension in this book.

**Note:**

The **ibm\_db2** extension API makes porting an application that was previously written for Unified ODBC almost as easy as globally changing "odbc\_" to "db2\_" throughout the source code of your application since all function names start with "odbc\_" in Unified ODBC and "db2\_" in **ibm\_db2** with the rest of the name being identical in both.

### 8.3.2 PHP development with the ibm\_db2 extension

If you are working in Linux/UNIX, before using **ibm\_db2**, make sure that the php.ini configuration has been set to the DB2 instance you want PHP to connect to, so that PHP will refer to the libraries of the respective instance for connecting and querying the

database. In Linux/UNIX, this overrides the environment variable DB2INSTANCE, while in Windows this option is simply ignored.

If not already present, you can make an entry in the `php.ini` file as follows assuming you are using the default instance name of `db2inst1`:

```
[ibm_db2]
ibm_db2.instance_name=db2inst1
```

Another global variable that we can change in the `php.ini` file is the `ibm_db2.binmode` which can be used to modify the binary data handling by the PHP driver. The syntax is:

```
ibm_db2.binmode = "n"
```

Where  $n = \{1,2,3\}$ .

When set to "1", the `DB2_BINARY` constant gets set and all binary data is handled as it is.

When set to "2", the `DB2_CONVERT` constant gets set and all the binary data is converted into ASCII by the `ibm_db2` driver.

When set to "3", the `DB2_PASSTHRU` constant gets set and all the binary data gets converted into a null value.

### 8.3.2.1 Program flow

The typical steps followed by a PHP program are:

1. Connect to the database.
2. Prepare and execute the statement.
3. Process the results.
4. Free the resources.

PHP uses handlers for connections and statements. Handlers are variables that are set after a connection or a statement execution, and are useful to establish a relationship between a connection and the statements being executed.

### 8.3.2.2 Connecting to a Database

There are two methods to connect to a DB2 database:

- Non persistent connection (`db2_connect`)
- Persistent database connection (`db2_pconnect`)

As the name suggests, the non persistent connection disconnects and frees up the connection resources after each `db2_close`, or when the connection resource is set to NULL, or the script ends. Performance can be impacted if database sessions are made

and freed too often. However, it is advisable to use a non persistent connection when you are doing **INSERT**, **UPDATE**, or **DELETE** like operations.

In the case of persistent connections, the connection resources are not freed up after a **db2\_close** or the script is exited. Whenever a new connection is requested, PHP tries to reuse the connection with the same credentials.

The syntax of the connection varies depending on whether the database has been cataloged in DB2 or not.

#### 8.3.2.2.1 Making a connection to a cataloged database

To make a connection to a cataloged database, the database alias name, user ID, and password of the database server are the required parameters in **db2\_connect**. *Listing 8.1* shows an example for connecting to the SAMPLE database.

```
<?php
$db_name = 'Sample';
$usr_name = 'db2inst1';
$password = '123456';
// For persistent connection, change db2_connect to db2_pconnect
(1) $conn_resource = db2_connect($db_name, $usr_name, $password);
    if ($conn_resource) {
        echo 'Connection to database succeeded.';
(2) db2_close($conn_resource);
    } else {
        echo 'Connection to database failed.';
        echo 'SQLSTATE value: ' . db2_conn_error();
        echo 'with Message: ' . db2_conn_errormsg();
    }
?>
```

#### Listing 8.1 – Connecting to a cataloged database

In the above listing:

1. `conn_resource` is the connection handler after the **db2\_connect()** method obtains a connection
2. The **db2\_close()** method is used to close/relinquish the connection resources we had acquired before.

#### 8.3.2.2.2 Making a connection to a non-cataloged database

To create a connection to a non-cataloged remote database we need to pass the details about the database server into a connection string and pass it as a parameter in the connection function. The connection string should be in the following format:

```
DRIVER={IBM DB2 ODBC DRIVER};DATABASE=database name;HOSTNAME=host
```

```
name;PORT=port;PROTOCOL=TCPIP;UID=user name;PWD=password;
```

### 8.3.2.3 Preparing and executing a statement

Before you prepare and execute SQL statements, you need to decide the following characteristics about the transaction:

- Type of cursor used
- How to catch the error
- Isolation level to use

#### 8.3.2.3.1 Type of cursor to be used

PHP with `ibm_db2` supports two kinds of cursors:

- **Forward only cursors**

This is the default cursor of a PHP application with `ibm_db2`. The cursor fetches the result set row by row in a unidirectional way. It is an ideal cursor when we want to perform read-only operations against the database.

- **Scrollable cursors**

The `ibm_db2` extension implements scrollable cursors using the keyset-driven scrollable cursor. This cursor can detect and make changes to the underlying data. When the cursor is opened, DB2 makes a keyset where it stores the keys, which are used to determine the order and the set of rows in the cursor. As the fetch operation proceeds, the cursor scrolls through the keys in the keyset to retrieve the most recent values in the database.

#### 8.3.2.3.2 How to catch an error

The application you write should be good enough to catch and explain all exceptions to the user, including the SQL and database errors. For that, the program should check the return values of the database functions and print the `SQLSTATE` and the error message if an error has occurred. Use `db2_stmt_error` and `db2_stmt_errormsg` to display the error details when an error occurs. *Listing 8.2* provides an example.

```
$stmt = db2_exec($conn_resource, $sql);  
if (!$stmt) {  
    echo 'SQLSTATE value: ' . db2_stmt_error();  
    echo 'with Message: ' . db2_stmt_errormsg();  
}
```

#### Listing 8.2 – Displaying error details

### 8.3.2.3.3 Isolation level to use

You can use two methods to change the isolation level in a PHP program:

1. Append the **WITH** clause in the SQL statement so that a particular SQL runs in the specified isolation level. *Listing 8.3* provides an example.

```
// With connection being made and connection resource
// is in $conn_resource
$sql = 'SELECT c_id FROM customer WITH UR';
$stmt = db2_exec($conn_resource, $sql);
```

#### Listing 8.3 – Set isolation level in PHP with `ibm_db2`

In the above example, the query is run using the **UR** isolation level.

2. Changing the **CURRENT ISOLATION** special register

To use a particular isolation level for the whole session, set the **CURRENT ISOLATION** special register to **UR**, **CS**, **RS**, or **RR**. The DB2 special register value overrides the default isolation level. It is a good practice to reset the isolation level to the default (**CS**) toward the end of the script. See *Listing 8.4* for an example.

```
// With connection being made and connection resource
// is in $conn_resource
(1) $currentiso = 'SET CURRENT ISOLATION LEVEL TO RR';
(2) $sql = 'SELECT c_id FROM customer';
(3) $stmt = db2_exec($conn_resource, $currentiso);
(4) $stmt = db2_exec($conn_resource, $sql);
// Execute other SQL statements
    $currentiso = 'SET CURRENT ISOLATION LEVEL TO CS';
    $stmt = db2_exec($conn_resource, $currentiso);
```

#### Listing 8.4 – Set isolation level in **CURRENT ISOLATION** special register

In the above listing:

- (1) `currentiso` is assigned the current isolation level to **RR**.
- (2) `sql` is assigned the SQL query to be executed.
- (3) `stmt` is assigned the return values from the `db2_exec()` method. The connection resource variable, and the isolation level need to be passed while executing the command.
- (4) `stmt` is assigned the return values from the `db2_exec()` method. The connection resource variable and the query to be executed need to be passed while executing the command.

### 8.3.2.3.4 Preparing and executing SQL statements

The concept of preparing and executing SQL statements in one or two different steps have been discussed in Chapter 1. In PHP with the `ibm_db2` extension, this is done as follows:

### Prepare and execute together

Doing a prepare and execute in one step involves only one function, but it does not give you optimized performance if the same query is executed more than once. Passing the SQL statement along with the connection resource to the function `db2_exec` will prepare and then execute the statement in one step. The one step process can be used with different cursor types:

- To execute and prepare with default cursors :

```
$sql = 'SELECT c_id FROM customer';  
$stmt = db2_exec($conn_resource, $sql);
```

- To execute and prepare with a different type of cursor parameter, you can use an optional parameter for changing the default forward-only cursor to the scrollable cursor. You need to pass `DB2_SCROLLABLE` as an associative array as the third parameter as shown below:

```
$stmt = db2_exec($conn_resource, $sql, array('cursor' =>  
DB2_SCROLLABLE));
```

### Prepare and then execute

This is the best way to execute SQL statements in terms of security and performance. The steps involved in the procedure are:

#### 1. Preparing the SQL statement

You can prepare an SQL statement with or without parameter markers by using the `db2_prepare` function. You can also specify which type of cursor to use while fetching the rows. For example, this SQL statement uses parameter markers:

```
SELECT c_name FROM CUSTOMERS WHERE c_id = ? and c_phone = ?
```

The values for the parameter markers ("?") can be supplied to the database engine to retrieve the results using the `db2_prepare` function.

#### 2. Bind the parameters

Use the `db2_bind_param` function to bind a PHP variable into the prepared SQL statement dynamically. It is more powerful than binding an array of variables in the `db2_execute` statement, because we can specify the parameter type, data type, precision, and scale of the variable that we bind with the prepared SQL statement. The parameter `DB2_PARAM_IN` is used for all statements, except when inserting large objects and using the `CALL` statement. Once the parameter is

bound, it is assigned to memory, and the prepared statement is now populated with those values, which were not given during the `db2_prepare`. You can assign the value of the parameter in PHP after the binding also. *Listing 8.6* shows an example of preparing, binding with two parameter markers (one of which is an integer, and the other is a character), and executing a statement.

```
<?php
    $database = 'sample';
    $user = '';
    $password = '';
    $conn = db2_connect($database, $user, $password);
    if ($conn) {
        $statement = 'SELECT c_id, c_name, c_email FROM db2inst1.customer
                    WHERE c_id > ? OR c_name NOT LIKE ?';
// prepare the SQL statement
        $stmt = db2_prepare($conn, $statement);
        $id = 100;
(1) db2_bind_param($stmt, 1, "id", DB2_PARAM_IN);
        db2_bind_param($stmt, 2, "name", DB2_PARAM_IN, DB2_CHAR);
        $name = 'MyName';
        $result = db2_execute($stmt);
(2) while ($object = db2_fetch_object($stmt)) {
// Iterate through results
        echo 'ID: ' . $object->C_ID;
        echo 'Name: ' . $object->C_NAME;
        echo 'Email: ' . $object->C_EMAIL;
        }
        if (!$result) {
(3) db2_rollback($conn);
        echo 'Execution failed. ';
        echo 'SQLSTATE value: ' . db2_stmt_error();
        echo 'with Message: ' . db2_stmt_errormsg();
        }
        db2_free_stmt($stmt);
        db2_close($conn);
    } else {
        echo 'Connection to database failed.';
        echo 'SQLSTATE value: ' . db2_conn_error();
        echo 'with Message: ' . db2_conn_errormsg();
    }
?>
```

**Listing 8.6 – Preparing, binding, and executing SQL statements**

In the above listing:

1. The `db2_bind_param` binds a PHP variable to an SQL statement parameter
2. The `db2_fetch_object` returns an object with properties representing columns in the fetched row so we have a variable object to hold it.
3. The `db2_rollback()` rolls back an in-progress transaction on the specified connection resource and begins a new transaction. PHP applications normally default to `AUTOCOMMIT` mode, so `db2_rollback()` normally has no effect unless `AUTOCOMMIT` has been turned off for the connection resource.

**Note:**

If the specified connection resource is a persistent connection, all transactions in progress for all applications using that persistent connection will be rolled back. For this reason, persistent connections are not recommended for use in applications that require transactions.

In addition, note the following before you bind the variables:

- The PHP variable name needs to be enclosed in double quotes (") and without the dollar sign (\$) "variable".
- Check for the position variable of the bound parameters. The indexing should start at 1.
- For variables other than `INTEGER` and `VARCHAR`, we recommend you use the data type specified: `DB2_BINARY`, `DB2_CHAR`, `DB2_DOUBLE`, or `DB2_LONG`.

Another example using the `DECIMAL` data type, requires you to use the parameter `DB2_LONG` in `db2_bind_param`. In the statement below, `amount` is a `DECIMAL` type variable.

```
// Here we bind a decimal (10,2) type variable amount
db2_bind_param($stmt, 1, "amount", DB2_PARAM_IN, DB2_LONG);
```

### 3. Executing the query

Once you prepare the query, bind the parameter in `db2_bind_param`. The statement resource is obtained after the query is prepared and binding is passed as the input to `db2_execute`, which executes the statement. You can find a sample program for this method in *Listing 8.6*.

Alternatively, pass the parameter as an array directly to `db2_execute`. In this case, the array variable also needs to be provided as the second parameter in addition to the ones from `db2_bind_param`.

Once the query is executed, you can use the statement resource to get the result set using one of the following functions:



1. `db2_fetch_array`
2. `db2_fetch_assoc`
3. `db2_fetch_both`
4. `db2_fetch_object`
5. `db2_fetch_row`

Some of these functions are discussed in more detail in the next section.

#### 8.3.2.4 Processing the results

When an SQL statement returns a result set, there are different ways to retrieve the rows. You can fetch the result set row by row into a data structure or use scrollable cursors to scroll through the result set and then fetch the required rows.

##### 8.3.2.4.1 Using `db2_fetch_array`

This function is used to fetch the rows of the result set data into an array indexed by the column number. It takes the statement resource as input and returns false when there are zero rows returned. *Listing 8.7* provides an example.

```
while ($row = db2_fetch_array($stmt)) {  
    printf("%d %s %s", $row[0], $row[1], $row[2]);  
}
```

#### Listing 8.7 – Using `db2_fetch_array` function

##### 8.3.2.4.2 Using `db2_fetch_assoc`

This function is used to fetch the result set data into an array indexed by the column name. It returns a false if there are zero rows returned. *Listing 8.8* provides an example.

```
$sql = 'SELECT * FROM customer WHERE c_id >= ?';  
$stmt = db2_prepare($conn_resource, $sql);  
if (!$stmt) {  
    echo 'The prepare failed. ' ;  
    echo 'SQLSTATE value: ' . db2_stmt_error();  
    echo 'with Message: ' . db2_stmt_errormsg();  
} else {  
    db2_bind_param($stmt, 1, "c_id", DB2_PARAM_IN);  
    $c_id = 100;  
    $result = db2_execute($stmt);  
    if (!$result) {  
        echo 'The execute failed. ' ;  
        echo 'SQLSTATE value: ' . db2_stmt_error();  
        echo 'with Message: ' . db2_stmt_errormsg();  
    }  
}
```

```
}  
while ($row = db2_fetch_assoc($stmt)) {  
    printf("%d %s %s ", $row['C_ID'], $row['C_NAME'], $row['C_EMAIL']);  
}
```

#### **Listing 8.8 – Using db2\_fetch\_assoc function**

Since the columns are indexed by the column names, the case of the column names does matter. The default case of the column can be overridden using the `DB2_ATTR_CASE` statement option which can have one of these values:

- `DB2_CASE_NATURAL`: Column names as returned by DB2.
- `DB2_CASE_LOWER`: Column names are forced to lower case.
- `DB2_CASE_UPPER`: Column names are forced to uppercase.

For example, if you would like to force column names to lower case, use:

```
$stmt = db2_prepare($conn_resource, $sql, array('DB2_ATTR_CASE' =>  
DB2_CASE_LOWER));
```

#### **8.3.2.4.3 Using db2\_fetch\_both**

This function returns an array which is indexed by both the column number and the column name. *Listing 8.9* provides an example.

```
while ($row = db2_fetch_both($stmt)) {  
    printf("%d %s %s", $row[0], $row['C_NAME'], $row['C_EMAIL']);  
}
```

#### **Listing 8.9 – Using db2\_fetch\_both function**

#### **8.3.2.4.4 Using db2\_fetch\_object**

This function can be used to return an object for each row fetched. Each property of the object will be the column returned. *Listing 8.10* provides an example.

```
while ($row = db2_fetch_object($stmt)) {  
    printf("%d %s %s", $row->C_ID, $row->C_NAME, $row->C_EMAIL);  
}
```

#### **Listing 8.10 – Using db2\_fetch\_object function**

#### **8.3.2.4.5 Using db2\_fetch\_row**

This function can be used to iterate through the result set or go to the specified row number when using scrollable cursor. *Listing 8.11* shows how to fetch all the rows in the result set using `db2_fetch_row` function.

```

while ($row = db2_fetch_row($stmt)) {
    printf(    "%d %d %s",
        db2_result($stmt, 0),
        db2_result($stmt, 1),
        db2_result($stmt, 2)
    );
}

```

**Listing 8.11 – Using db2\_fetch\_row function**

#### 8.3.2.4.6 Retrieving metadata

There are functions which can be used to retrieve the metadata about the columns, which are returned by the result set. Some useful ones are:

- **db2\_num\_rows** returns the number of rows that were deleted, inserted, or updated by the SQL statement or the number of rows returned by the SQL statement. However, if you are using scrollable cursors to perform the SQL operation, the use of **db2\_num\_rows** should be avoided, and you should try to use the `'SELECT COUNT(*) FROM table WHERE condition'`.
- **db2\_fetch\_** function returns boolean values which can be used to test the end of a result set.
- Metadata functions with prefix **db2\_field\_** can be used to derive all types of information about the fields returned by an SQL statement. The information can be about column name, data type, precision, and scale for decimal type and width. These functions, if run successfully, return a string containing the information. The parameters needed for the functions are statement resource and the column number starting from 0 or the column name.
- **db2\_num\_fields** helps to get the number of columns returned by the **SELECT** statement. It is used to find out the number of columns returned by the dynamically generated queries and stored procedures. *Listing 8.12* provides an example.

```

$sql = 'SELECT * FROM customer WHERE c_id > ?';
$stmt = db2_prepare($conn_resource, $sql);
// The number of columns in the statement
echo 'Num fields: ' . db2_num_fields($stmt) . ' in the statement';
if (!$stmt) {
    echo 'The prepare failed. ';
    echo 'SQLSTATE value: ' . db2_stmt_error();
    echo 'with Message: ' . db2_stmt_errormsg();
} else {
    db2_bind_param($stmt, 1, "c_id", DB2_PARAM_IN);
    $c_id = 100;
}

```

```
    $result = db2_execute($stmt);
if (!$result) {
    echo 'The execute failed. ';
    echo 'SQLSTATE value: ' . db2_stmt_error();
    echo 'with Message: ' . db2_stmt_errormsg();
}
}
```

**Listing 8.12 – Using metadata functions****8.3.2.5 Freeing the resources**

You can free statement resources using `db2_free_stmt` and result set resources using `db2_free_result`. If you do not use these functions to free the resources, both resources are automatically freed after the script finishes.

The DB2 function `db2_close` can be used to free the connection resources allocated by `db2_connect`. If `db2_pconnect` is used to establish persistent connections, the `db2_close` request is ignored and the connection resource is used by the next similar connection. *Listing 8.13* provides an example of using `db2_close`.

```
$result = db2_close($conn_resource);
if ($result) {
    echo 'Connection was successfully closed.';
}
}
```

**Listing 8.13 – Using the `db2_close()` method****8.3.3 PHP development with PDO\_IBM/PDO\_ODBC**

This section describes PHP development using the `pdo_ibm` driver. Other than a few differences in configuration parameter names, PHP development with `pdo_ibm` and `pdo_odbc` is exactly the same. After the installation and configuration of DB2 and these drivers along with PHP and Apache, there are some variables that may need to be edited in the `php.ini` file to make PDO work with DB2. Some of these variables were discussed earlier, but more explanation is provided below:

- **`pdo_odbc.db2_instance_name`**

This entry has to be included in Linux and UNIX installations in order to guide PHP to the instance where the DB2 libraries are located. Below are some examples assuming your instance name is `db2inst1`.

For `pdo_ibm` use:

```
[PDO_IBM instance name]
pdo_ibm.db2_instance_name= db2inst1
```

For PDO\_ODBC use:

```
[PDO_ODBC instance name]
pdo_odbc.db2_instance_name= db2inst1
```

- **pdo\_odbc.connection\_pooling**

*Connection pooling* helps database application performance by reusing old connections established before. It is especially beneficial when the connections made have a short duration. The value of `pdo_odbc.connection_pooling` can be configured so as to decide the level of the connection pooling. The different levels are listed in *Table 8.1*.

Connection pooling level	Description
Strict	A connection is reused when the connection credentials exactly match the connection options of an existing closed connection. We recommend to use this option when connection pooling is enabled.
Relaxed	Connections with a matching connection string are reused and not all connection attributes are checked.
Off	Connection pooling is not used

The following example shows an entry in `php.ini` when the connection pooling is set to relaxed:

```
[PDO connection pooling]
pdo_ibm.connection_pooling = relaxed
or
[PDO connection pooling]
pdo_odbc.connection_pooling = relaxed
```

- **pdo.dsn.\***

Data Source Name (DSN) contains the information related to the connectivity to a database through an ODBC or CLI driver. It contains the database name, database driver, user name, password, and other information. You can make an alias for the DSN string in the `php.ini`. To create an alias, make an entry for each alias starting with `pdo.dsn` followed by the name of the alias that is equated to the DSN. The DSN in this scenario can be for both, cataloged and non-cataloged DB2 databases. The following example shows the entry for the DSN alias assuming you want to connect to the SAMPLE database:

```
For pdo_ibm:  
[DSN alias for pdo_db2]  
pdo.dsn.dealerbase="ibm:sample"
```

```
For pdo_odbc:  
[DSN alias for pdo_db2]  
pdo.dsn.dealerbase="odbc:sample"
```

Once the entry has been made in the `php.ini` file, we can connect to the databases referring to the alias name as shown below:

```
$dbh = new PDO("dealerbase");
```

### 8.3.3.1 Program flow

PDO programming uses object-oriented concepts. When a connection is made to a DB2 database, an instance of the PDO base class is created. This instance acts as a database handle, which is later used when further activities are carried on against the database. Different objects are created when operating with a database, and can be classified as:

- Connection object (instance of `PDO`)
- Statement object (instance of `PDOStatement`)
- Exception (instance of `PDOException`)

The typical program flow of a PDO program during transaction processing is:

1. Connect to the database.
2. Prepare and execute the statement.
3. Process the results.
4. Free the resources.

### 8.3.3.2 Connecting to a database

A connection to a database is obtained when the constructor of the PDO class is invoked using `new PDO()`. There are four sets of parameters for the constructor:

- Data Source Name (DSN)
- User name
- Password
- Driver options

DSN contains the information required to connect to a database, which includes the type of driver used (CLI/ODBC in the case of DB2), the name of the data source, and the connection credentials (the user name and password). The DSN parameters required depend on whether you are connecting to a cataloged database, or a non-cataloged one. If

you are connecting to a DB2 database, it is better to use cataloged connections. Below are examples of how to specify the DSN assuming the database name is `SAMPLE`, the user ID is `db2inst1`, and the password is `123`:

- DSN for cataloged connection

For `pdo_ibm`:

```
ibm:DSN=sample;UID=db2inst1;PWD=123
```

For `pdo_odbc`:

```
odbc:DSN=sample;UID=db2inst1;PWD=123
```

- DSN for non-cataloged connection

For `pdo_ibm`:

```
ibm:DSN={IBM DB2 ODBC
DRIVER};HOSTNAME=localhost;PORT=50000;DATABASE=sample;PROTOCOL=TCPIP
;UID=db2inst1;PWD=123;
```

For `pdo_odbc`:

```
odbc:DSN={IBM DB2 ODBC
DRIVER};HOSTNAME=localhost;PORT=50000;DATABASE=sample;PROTOCOL=TCPIP
;UID=db2inst1;PWD=123;
```

An example of a connection using the cataloged method is provided in *Listing 8.14* below.

```
<?php
try {
/* Use one of the following connection string */
/* For PDO_IBM */
$constrng = 'ibm:DSN=sample;UID=db2inst1;PWD=123';
/* for PDO_ODBC */
$constrng = 'odbc:DSN=sample;UID=db2inst1;PWD=123';
$dbh = new PDO($constrng);
echo 'Connected';
} catch (PDOException $exp) {
echo 'Exception: ' . $exp->getMessage();
}
?>
```

#### **Listing 8.14 – Cataloged connection to a database**

In *Listing 8.14*, the new `PDO()` method establishes a new connection.

To write a program to connect to the database, using non-cataloged connection, follow the same technique as discussed in section 8.3.2.2.2 earlier.

The DSN name can be provided to a PDO program in three ways:

- **As a parameter to the constructor.**

Using this method, the DSN is provided as a string in the program. However, this makes the program dependent on the database. Every time the database location or password is changed, the corresponding entry has to be changed in all programs referring to this database. Listing 8.15 provides an example using this method.

```
<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:sample';
/* The connection below is for pdo_odbc */
$constring = 'odbc:sample';
try {
$dbh = new PDO($constring, '', '');
echo 'Connected';
} catch (PDOException $exp) {
echo 'Exception: ' . $exp->getMessage();
}
?>
```

**Listing 8.15 – DSN as a parameter to the constructor**

▪ **In a file**

Using this method, the DSN is provided in a file and the file is referenced inside the program. If the DSN parameter value changes, only the file has to be changed. You do not need to change the program every time the DSN is changed. For example, in *Listing 8.16* below, assume you have created a file named `dsnfile` and stored it in `/usr/local` directory.

```
<?php
$constring = 'uri:file:///usr/local/dsnfile';
try {
$dbh = new PDO($constring, '', '');
echo 'Connected';
} catch (PDOException $exp) {
echo 'Exception: ' . $exp->getMessage();
}?>
```

**Listing 8.16 – Creating and storing a file**

▪ **Using aliasing in `php.ini`**

Similar to the `ibm_db2` extension, you can use both persistent and non-persistent connections in `PDO_IBM/PDO_ODBC`. If connection pooling is enabled by setting the `pdo_odbc.connection_pooling` to either `strict` or `relaxed`, we should not be making persistent connections, since PDO will not free the connection to the ODBC layer for the connection pooling to happen. It will be better to use connection pooling over persistent connections. To make a non-persistent connection, we



create an instance of PDO using the DSN, user name, and password as shown in *Listing 8.17* below.

```
<?php
/* Use one of the following connection strings */
/* The connection below is for pdo_ibm */
$constring = 'ibm:sample';
/* The connection below is for pdo_odbc */
$constring = 'odbc:sample';
try {

$dbh = new PDO($constrng, 'db2inst1', '123');
echo 'Connected';
} catch (PDOException $exp) {
echo 'Exception: ' . $exp->getMessage();
}
?>
```

#### **Listing 8.17 – Making a Non-persistent connection**

To make a persistent connection use the following `new PDO` format:

```
$dbh = new PDO($constrng, 'db2inst1', '123',
array(PDO::ATTR_PERSISTENT=> true));
```

### **8.3.3.3 Preparing and executing the statement**

Before you prepare and execute SQL statements, you need to decide the following characteristics about the transaction:

- Type of cursor used
- How to catch the error
- Isolation level to use

#### **8.3.3.3.1 Type of cursor to be used**

Like the `ibm_db2` extension, PDO also supports two kinds of cursors:

##### **Forward-only cursor**

This cursor is the default cursor in the PDO driver. It is the fastest cursor available. Make sure that the cursor is closed using the method `PDOStatement::closeCursor` after fetching all the rows and before launching another query. The Forward-only cursor can be set using `PDO::ATTR_CURSOR` with the value of `PDO::CURSOR_FWDONLY` in the `PDOStatement::prepare`. *Listing 8.20* provides an example.

```
$sql = "SELECT c_id, c_name, c_email FROM customer WHERE c_name =
```

```

:name";
try {
$sth = $dbh->prepare($sql, array(PDO::ATTR_CURSOR,
PDO::CURSOR_FWDONLY));
$sth->execute(array(':name' => 'Myname'));
print_r($sth->fetchAll());
} catch (PDOException $exp) {
echo 'Exception: ' . $exp->getMessage();
}
}

```

### Listing 8.20 – Using Forward only cursor

#### Scrollable cursor

We can specify the scrollable cursor by setting the `PDO::ATTR_CURSOR` to `PDO::CURSOR_SCROLL`. *Listing 8.21* provides an example.

```

$sql = "SELECT c_id, c_email FROM customer WHERE c_name = :name";
try {
$sth = $dbh->prepare($sql, array(PDO::ATTR_CURSOR,
PDO::CURSOR_SCROLL));
$sth->execute(array(':name' => 'Daniel'));
while ($row = $sth->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_NEXT)) {
$data = $row[0] . "\t" . $row[1] . "\t" . $row[2] . "\n";
echo $data;
}
$sth->closeCursor();
} catch (PDOException $exp) {
echo 'Exception: ' . $exp->getMessage();
}
}

```

### Listing 8.21 – Using a scrollable cursor

#### 8.3.3.3.2 Catching errors in PDO

PDO has **exception handlers** to catch an exception and display its details. These handlers have a “try and catch” construct similar to other programming languages.

When an exception is raised by a PDO program, an instance of `PDOException` is created. The `PDOException` class is an extension of the `Exception` class and has three modes of operation. The error handling modes are forced by setting the PDO attribute `PDO::ATTR_ERRMODE`. *Table 8.2* describes the values it can take.

Value	Description
<code>PDO::ERRMODE_SILENT</code>	This is the default mode of error handling in PDO. The respective error code and

	messages are set in both the statement object ( <b>PDOStatement</b> ) and the database object ( <b>PDO</b> ) when an error happens. For statements using this mode, the error is not caught and the next statement is executed after the function fails.
PDO::ERRMODE_WARNING	This mode is useful when you are developing or testing your PDO application. Along with the setting of the error code and message, it raises an <b>E_WARNING</b> .  You can suppress this by using a "@" in front of the function name.
PDO::ERRMODE_EXCEPTION	This is the best way to handle errors in a PDO program. Once the error happens, an exception is thrown which prevents the execution of later functions in the try block. The control goes to the catch block where the exception is handled. Error information and a stack trace are provided.

**Table 8.2 - Error handling modes set in parameter PDO::ATTR\_ERRMODE**

*Listing 8.18* below provides an example of setting the error handling mode to PDO::ERRMODE\_EXCEPTION and how an exception is handled using a try and catch block. The program purposely issues an incorrect SQL statement that uses a column that does not exist in the table. When the error happens, the **PDOexception** is thrown and the details of the exception object is printed using the **getMessage()** method.

```
<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:sample';
/* The connection below is for pdo_odbc */
$constring = 'odbc:sample';
$dbh = new PDO($constring, 'db2inst1', '123');
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$sql = "SELECT c_id FROM customer WHERE notexistingcol = :name ";
$stmt = $dbh->prepare($sql);
try {
$stmt->execute(array(':name' => 'ABC'));
$row = $stmt->fetchAll();
print_r($row);
```

```
$sth->closeCursor();
} catch (PDOException $exp) {
echo 'Exception: ' . $exp->getMessage();
}
?>
```

#### **Listing 8.18 – Exception handling using PDO::ERRMODE\_EXCEPTION**

When the exception is generated, the `getMessage()` method produces an output that looks like this:

```
Exception : SQLSTATE[42S22]: Column not found: -206 [IBM][CLI
Driver][DB2/LINUX] SQL0206N "NOTEXISTINGCOL" is not valid in the context
where it is used. SQLSTATE=42703 (SQLExecute[-206] at /root/Desktop/php-
5.1.2/ext/pdo_odbc/odbc_stmt.c:133)
```

The methods called `errorCode()` and `errorInfo()` could be used to get more information about the error:

**PDO::errorCode()** gives the information about the **SQLSTATE** of the error.

**PDO::errorInfo()** gives all the information about the error, such as **SQLSTATE**, **SQLCODE**, and error message from the database server.

#### **8.3.3.3 Isolation level to use**

Considerations of which isolation level to use in PHP with PDO\_IBM/PDO\_ODBC are the same as the ones described for the `ibm_db2` extension earlier.

#### **8.3.3.3.4 Preparing and executing SQL statements**

PDO provides mechanisms by which you can prepare and execute SQL statements in a single step, and also in separate steps.

##### **Preparing and executing SQL statements in a single step**

Normally, we use this method for execution of SQL statements which are not executed repeatedly in the same program. We have two kinds of SQL statements:

- **SQL statements returning result sets**

The function `PDO::query` is used for preparing and executing the SQL statement which returns a result set indexed by both column position and column name. *Listing 8.19* is an example for the query function. You see that each row in the result set returned by the query is fetched into an array named `$row` and the value is printed.

```
<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
```

```

$constring = 'ibm:sample';
/* The connection below is for pdo_odbc */
$constring = 'odbc:sample';
$dbh = new PDO($constring, 'db2inst1', '123');
$sql = 'SELECT c_id, c_name FROM customer';
try {
    foreach ($dbh->query($sql) as $row) {
        echo $row[0] . ' ' . $row['C_NAME'];
    }
} catch (PDOException $exp) {
    print_r($sth->errorInfo());
    echo 'Exception: ' . $exp->getMessage();
}
?>

```

**Listing 8.19 – Prepare and execute SQL returning result set**

- **SQL statements not returning result sets**

PDO has a different way of preparing and executing SQL queries which do not return a result set. `PDO::exec` is used for this type of SQL statement. `PDO::exec` returns the number of rows affected by the function. The SQL statements, such as **DELETE**, **INSERT**, and **UPDATE** are executed using `PDO::exec`, and it returns the number of rows affected by this function.

```

<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:sample';
/* The connection below is for pdo_odbc */
$constring = 'odbc:sample';
$dbh = new PDO($constring, 'db2inst1', '123');
$num = $dbh->exec("INSERT INTO customer (c_name, c_email) VALUES
('Piotr',
'aa123@123.com')");
echo 'Number of rows affected is: ' . $num;
?>

```

**Listing 8.20 – Preparing and executing SQL that does not return a result set**

### Preparing and executing SQL statements in separate steps

If you plan to execute the same SQL multiple times with different parameters, it is better to prepare once, and execute many times. It is better for performance, and is more secure when compared to the combined preparation and execution method, since it checks for the data type every time a new parameter is bound against the database, therefore, avoiding situations such as SQL injection.

Parameter markers are of two types in PDO:

- Named parameter markers

You can give a name to the parameter marker with ":" prefixed to the name of the parameter. This parameter marker value is bound to the SQL to complete and then execute it.

- Nameless parameter markers

The ? symbol is used in the SQL so that DB2 understands that the values for this parameter marker will be bound later.

You cannot use both named and nameless parameter markers in the same statement.

There are three steps involved in preparing and executing an SQL statement:

### 1. Preparing the SQL statement

The PDO function `PDO::prepare` is used to prepare an SQL statement with or without parameter markers. In this function, the option of selecting which cursor to use is done using the driver option. You can also set this option later using the `setAttribute` method. For example, to set the cursor to scrollable while preparing the statement, you can use:

```
$sth = $dbh->prepare($sql, array(PDO::ATTR_CURSOR,
PDO::CURSOR_SCROLL));
```

The default cursor in PDO\_IBM/PDO\_ODBC is forward only. If you have set the cursor to scrollable, you change the cursor back to forward only by setting the attribute `PDO::ATTR_CURSOR` to `PDO::CURSOR_FWDONLY`.

### 2. Binding the parameters

There are two ways to bind parameters of an SQL statement:

#### Using the bind function

You can use either `PDOStatement::bindParam` or `PDOStatement::bindValue` for this task. These two methods can be called for the instance of `PDOStatement`, which we get after preparing the SQL statement using `PDO::prepare`. We need to specify the type of data which is bound using the function. The types are:

- `PDO::PARAM_INT` for integer data type
- `PDO::PARAM_STR` for string data type
- `PDO::PARAM_LOB` for large objects
- `PDO::PARAM_NULL` for binding a null value

*Listing 8.21* shows how to use `bindParam` and `bindValue` along with named parameter markers.

```
<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:sample';
/* The connection below is for pdo_odbc */
$constring = 'odbc:sample';
$dbh = new PDO($constring, '', '');
$cid = 100;
$name = 'ABC';
$sql = "SELECT c_id, c_name FROM customer WHERE c_id > :id AND c_name NOT
LIKE :name";
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
try {
    $sth = $dbh->prepare($sql);
    $sth->bindValue(':id', $cid, PDO::PARAM_INT);
    $sth->bindParam(':name', $name, PDO::PARAM_STR, 10);
    $sth->execute();
    $result = $sth->fetchAll();
    print_r ($result);
} catch (PDOException $exp) {
    print_r($sth->errorInfo());
    echo 'Exception: ' . $exp->getMessage();
}
?>
```

### Listing 8.21 – Using the bind function

We can have an additional length parameter which can force the length of the parameter which is bound using the `PDOStatement::bindParam`.

### Binding by passing parameter values in an array

*Listing 8.22* is an example for using the nameless parameter marker in a PDO program.

```
<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:sample';
/* The connection below is for pdo_odbc */
$constring = 'odbc:sample';
$dbh = new PDO($constring, '', '');
$cid = 100;
$name = 'ABC';
```

```
$sql = "SELECT c_id, c_name FROM customer WHERE c_id > ? AND c_name NOT
LIKE ?";
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
try {
    $sth = $dbh->prepare($sql);
    $rc = $sth->bindValue(1, $cid, PDO::PARAM_INT);
    $rc = $sth->bindParam(2, $name, PDO::PARAM_STR, 10);
    $rc = $sth->execute();
    $result = $sth->fetchAll();
    print_r($result);
} catch (PDOException $exp) {
    print_r($sth->errorInfo());
    echo 'Exception: ' . $exp->getMessage();
}
?>
```

**Listing 8.22 – Binding by passing parameter values in an array**

### 3. Executing the statement

The ***PDOStatement::execute*** is used to execute a prepared statement. The variables can be bound using a binding function, or the statement could be bound along with execution by passing a parameter as an array to this function. Listing 8.23 shows how to use a named parameter marker in the execute function.

```
<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:sample';
/* The connection below is for pdo_odbc */
$constring = 'odbc:sample';
$dbh = new PDO($constring, '', '');
$id = 10;
$sql = "select c_name from customer where c_id > :id";
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
try {
    $sth = $dbh->prepare($sql);
    $rc = $sth->execute(array(':id' => $id));
    $result = $sth->fetchAll();
    print_r($result);
} catch (PDOException $exp) {
    print_r($sth->errorInfo());
    echo 'Exception: ' . $exp->getMessage();
}
```



?>

### Listing 8.23 – Using PDOStatement::execute with a named parameter marker

#### 8.3.3.3.5 Creating and calling stored procedures

You can create SQL stored procedures using PHP with PDO just like you execute any other SQL. But if you need to call the stored procedure from the client, you need to take care of binding the parameters depending on their type. There are three types of parameters in a stored procedure and they are **IN**, **OUT**, and **INOUT**. In the bind parameter, the data type constant and the parameter are associated with bitwise OR operator (**|**). The parameter type **PDO::PARAM\_INPUT\_OUTPUT** is used for the **OUT** and **INOUT** parameters. No parameter type is needed for the **IN** parameter. *Listing 8.24* shows a stored procedure with all three types of parameters and returning result sets. Once the stored procedure is executed, the variables which are bound with the stored procedure parameter markers will be populated with the values from the stored procedure execution.

```
<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:sample';
/* The connection below is for pdo_odbc */
$constring = 'odbc:sample';
$dbh = new PDO($constring, 'db2inst1', '123');
$id = 10;
$sql = 'CALL getdetails(?, ?, ?)';
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$amount = 0;
$c_id = 140;
$quantity = 100;
try {
    $sth = $dbh->prepare($sql);
    // Input parameter
    $sth->bindParam(1, $c_id, PDO::PARAM_INT);
    // Output parameter
    $sth->bindParam(2, $amount, PDO::PARAM_INT|PDO::PARAM_INPUT_OUTPUT);
    // Input output parameter
    $sth->bindParam(3, $quant, PDO::PARAM_INT|PDO::PARAM_INPUT_OUTPUT);
    $sth->execute();
    echo 'The INOUT parameter is: ' . $quant . ' and the OUT parameter is: ' .
    $amount;
    $rowset = $sth->fetchAll(PDO::FETCH_NUM);
    print_r($rowset);
} catch (PDOException $exp) {
```

```
print_r($sth->errorInfo());
echo 'Exception: ' . $exp->getMessage();
}
?>
```

**Listing 8.24 – Calling stored procedure in PHP with PDO\_IBM/PDO\_ODBC****8.3.3.3.6 Handling transactions**

PDO, by default, runs in **autocommit** mode. That is, all queries are either committed (if successful) or rolled back (if unsuccessful). You can use methods like **PDO::beginTransaction** to make multiple SQL queries be a part of a single transaction. This function turns off the autocommit mode of the PDO application. Once the transaction completes, it can be committed using the **PDO::commit** function or rolled back using **PDO::rollback**.

*Listing 8.25* shows PHP handling a transaction with DB2 PDO\_IBM or PDO\_ODBC. You can see that when an error happens, the exception is thrown as the error handling mode is set to **PDO::ERRMODE\_EXCEPTION**. In the catch block, the transaction is rolled back and the error is printed. If no error occurs, the transaction is committed and completed.

```
<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:sample';
/* The connection below is for pdo_odbc */
$constring = 'odbc:sample';
$dbh = new PDO($constring, 'db2inst1', '123');
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
try {
$dbh->beginTransaction();
$dbh->exec("
INSERT INTO customer (c_name, c_email) VALUES ('ABC', 'a123@123.com')
");
$dbh->exec("
INSERT INTO customer (c_name, c_email) VALUES ('DEF', '1a23@123.com')
");
$dbh->exec("
INSERT INTO customer (c_name, c_email) VALUES ('GHI', '1n23@123.com')
");
$dbh->commit();
} catch (PDOException $exp) {
print 'Rolling back transaction';
$dbh->rollback();
print_r($dbh->errorInfo());
```

```

echo 'Exception: ' . $exp->getMessage();
}
?>

```

### Listing 8.25 – PHP with IBM\_PDO/PDO\_ODBC

#### 8.3.3.4 Processing the results

PDO offers two ways to fetch data:

##### 1. Buffered fetch

Buffered fetch allows you to fetch all the rows in the result set returned by an SQL query into an array. If the query is to return a huge result set, we do not recommend to use this method since it will consume a lot of system resources.

##### 2. Non-buffered fetch

Using `PDOStatement::fetch`, you can fetch each row at a time and manipulate the result set. The `PDOStatement::fetch` has different options to fetch the column for different data. The following sections describe these options.

#### Using FETCH\_BOUND and bindColumn

This method allows each column returned by the result set to be assigned to a variable. After performing `PDOStatement::fetch`, each of the corresponding values of the variable which was bound is updated to the value of the result set. The constant `PDO::FETCH_BOUND` is a parameter for the fetch, so that the bound column variables are populated by the fetch function. *Listing 8.26* shows an example for the fetch of bound columns.

```

<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:sample';
/* The connection below is for pdo_odbc */
$constring = 'odbc:sample';
$dbh = new PDO($constring, '', '');
$id = 10;
$sql = "select c_id, c_name from customer where c_id > :id";
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
try {
    $sth = $dbh->prepare($sql);
    $sth->execute(array(':id' => $id));
    $sth->bindColumn(1, $id);
    $sth->bindColumn(2, $name);
}

```

```
while ($row = $sth->fetch(PDO::FETCH_BOUND)) {
    echo 'id is: ' . $id . ' and name is: ' . $name;
}
} catch (PDOException $exp) {
    print_r($sth->errorInfo());
    echo 'Exception: ' . $exp->getMessage();
}
?>
```

#### **Listing 8.26 – Fetching data using *FETCH\_BOUND***

#### **Using the Fetch function with no parameters passed**

This is the default option. The fetch function returns each row of the result set with the columns indexed by its position. *Listing 8.27* provides an example.

```
while ($row = $sth->fetch()) {
    echo 'id is: ' . $row[0] . ' and name is: ' . $row[1];
}
```

#### **Listing 8.27 – Example using the Fetch function**

#### **Using *FETCH\_ASSOC***

We can fetch the rows of the result set into an array indexed by the column name of the result set, which is returned by the query. *Listing 8.28* provides an example.

```
while ($row = $sth->fetch(PDO::FETCH_ASSOC)) {
    echo 'id is: ' . $row['C_ID'] . ' and name is: ' . $row['C_NAME'];
}
```

#### **Listing 8.28 – Example using the *Fetch\_Assoc* function**

#### **Using *FETCH\_BOTH***

This fetch with parameter `PDO::FETCH_BOTH` passed a parameter to `PDOStatement::fetch` and will return an array for each row indexed by both column names and the position. *Listing 8.29* provides an example.

```
while ( $row = $sth->fetch(PDO::FETCH_BOTH)) {
    echo 'id is: ' . $row[0] . ' and name is: ' . $row['C_NAME'];
}
```

#### **Listing 8.29 – Example using *Fetch\_Both***

#### **Using scrollable cursors**

We can scroll through the cursor if we are using scrollable cursor and passing the cursor `PDO::FETCH_ORI_NEXT` constant to fetch the next row from the result set. *Listing 8.30* provides an example.

```
$sql = "SELECT * FROM customer";
try {
    $offset = 0;
    $sth = $dbh->prepare($sql, array(PDO::ATTR_CURSOR, PDO::CURSOR_SCROLL));
    $sth->execute();
    while ($row = $sth->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_NEXT)) {
        $data = $row[0] . "\t" . $row[1] . "\t" . $row[2] . "\n";
        echo $data;
    }
    $sth->closeCursor();
} catch (PDOException $exp) {
    print_r($sth->errorInfo());
    echo 'Exception: ' . $exp->getMessage();
}
```

#### Listing 8.30 – Using scrollable cursors

#### Using fetch into objects

You can fetch the data directly into objects by providing the constants `PDO::FETCH_LAZY`, and `PDO::FETCH_OBJ`. In `PDO::FETCH_LAZY`, the object variable name that is used to access the columns is made as they are accessed. We can refer to the columns both by column names or the respective positions. *Listing 8.31* provides an example.

```
while ($row = $sth->fetch(PDO::FETCH_LAZY)) {
    print_r($row->C_NAME);
}
while ($row = $sth->fetch(PDO::FETCH_OBJ)) {
    print_r($row->C_NAME);
}
```

#### Listing 8.31 – Using Fetch to refer columns

#### 8.3.3.5 Freeing the resources

You need to close the cursor every time a fetch is done so that the statement resource can be reused. The method used to close the cursor is `PDOStatement::closeCursor`. The connection resource can be freed by assigning it to a null value. An example is provided in *Listing 8.32*.

```
// Connect to the database, use one of the following
// for PDO_IBM
$dbh = new PDO('ibm:sample', '', '');
```

```
// for PDO_ODBC
$dbh = new PDO('odbc:sample', '', '');
// Perform database operations here
...
// Free the connection
$dbh = null;
```

**Listing 8.32 – Freeing the connection resource**

## 8.4 Optimizing DB2 usage with PHP

Once your PHP application is up and running, you can tune both your application and DB2 to get the best performance. DB2 has an effective cost-based optimizer to make sure that each query you write gets executed in the best possible manner depending on your environment.

### 8.4.1 Design considerations for increasing the PHP-DB2 performance

The following points need to be considered while designing and coding you application:

- Use the `prepare` and then `execute` method for executing SQL statements which are repeatedly executed.
- Use connection pooling in PDO driver and persistent connection in the `ibm_db2` driver or enable connection concentrator in DB2.
- Commit often as long as your logic allows you to. This releases locks allowing for more concurrency.
- Push more logic to the database layer and use user defined functions, stored procedures, and SQL/XML, to reduce network traffic and save computing resources.
- Make optimal use of the storage space available; store large tables, in table spaces spread over different disks, and store DB2 logs in different disks.
- Index your table columns effectively using the DB2 Design Advisor.
- Change your database and database manager configuration to optimize db2 for your environment using the Configuration Advisor wizard.
- `RUNSTATS` regularly on all tables along with system catalog tables.
- Make your buffer pool use 75% of available memory if possible.
- Use monitoring tools to monitor the DB2 activity.
- Use the least restrictive isolation level that maintains the data integrity requirements of the application.

## 8.5 Exercises

In this exercise, we will practice writing a small PHP script to access data in the **SAMPLE** database.

Before running the exercise, login onto the server as the instance owner (for example *db2inst1* on Linux or *db2admin* on Windows), and then:

1. Run the following command to create the **SAMPLE** database if you haven't done so before: `db2samp1`
2. Write a PHP script to insert and retrieve data from the table EMPLOYEE.
3. To Run the PHP script in the Apache web server, on Windows, copy the PHP file, `db2employee.php`, to the `C:/Program Files/Apache Group/Apache2/htdocs` directory.
4. The sample script for the exercise is provided in the script `db2employee.php` accompanying the book. You can modify the user ID, password and host in the script appropriately and test it by using the following URL to run this PHP script with a Web browser: `http://localhost/db2employee.php`

## 8.6 Summary

In this chapter, we studied how to setup an environment for PHP applications to run on DB2. You learned some basic concepts involving PHP and DB2 application development which included the usage of the `ibm_db2` extension and the `pdo_ibm/pdo_odbc` driver to prepare and execute SQL on DB2 through PHP.

## 8.7 Review questions

1. What is the difference between freeing of resources using `db2_connect` and `db2_pconnect`?
2. What is connection pooling? When is it beneficial?
3. How do you enable PHP support in Apache HTTP Server 2.x?
4. How do you determine the configuration file path for your PHP installation?
5. What steps need to be followed to port an application previously written for Unified ODBC to `ibm_db2`?
6. Below are different FETCH methods used with the `ibm_db2` extension EXCEPT:
  - A. `db2_fetch_array`
  - B. `db2_fetch_assoc`
  - C. `db2_fetch_both`
  - D. `db2_fetch_column`

- E. None of the above
7. What is ibm\_db2?
- A. A driver
  - B. An extension
  - C. An adapter
  - D. A data object
  - E. None of the above
8. SQL stored procedures using PDO use which of the following type of parameter?
- A. INPUT
  - B. OUTPUT
  - C. EXECUTE
  - D. INOUT
  - E. None of the above.
9. Which of the following files hold the configuration details in PHP?
- A. httpd.conf
  - B. php\_conf
  - C. php.ini
  - D. httpd.ini
  - E. None of the above
10. Which is the best way to execute SQL statements in terms of security and performance?
- A. Prepare and then execute
  - B. Execute
  - C. Prepare
  - D. Prepare and execute together
  - E. None of the above



# 9

## Chapter 9 – Application development with Perl

*Perl (Practical Extraction and Reporting Language)* is a platform independent, interpreted scripting language free and available with many operating systems, such as Windows, Linux, UNIX and Mac OS X. As an open source language, the Perl interpreter can be freely downloaded in source code or binary form.

Because of its powerful text processing features, for example its strong support of regular expressions, Perl has been widely used not only for report generating applications, but also for standalone or Web-based database applications. You can easily generate dynamic SQL queries, and fetch and process the returned data using Perl scripts.

In this chapter you will learn about:

- Setting up the Perl environment to work with DB2
- Connecting to a DB2 database from a Perl application
- Developing Perl applications to access DB2 data

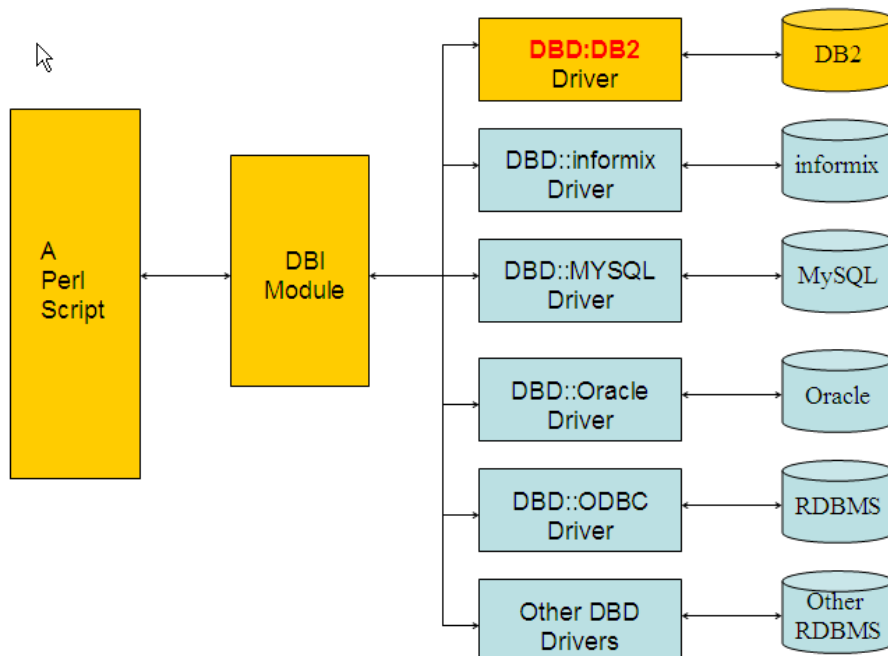
**Note:**

If you are new to the Perl programming language, review the free ebook [Getting Started with Perl](#) which is part of this book series.

### 9.1 Perl - DB2 applications: The big picture

A Perl application uses the *Perl Database Interface (DBI)* to access most of relational databases, including DB2. The Perl DBI is an open standard application programming interface (API) that enables a Perl application to access different databases using the same syntax without knowing details of each specific database. These details are handled by the underlying *DataBase Drivers (DBD)*, which are provided by the database vendors supporting the Perl DBI.

*Figure 9.1* illustrates how a Perl script accesses different databases via the DBI module and the DBD drivers.



**Figure 9.1 - A Perl script uses DBI and DBD drivers to access databases**

As shown in *Figure 9.1*, the DBI module provides a database independent interface standing between a Perl application and one or more database driver modules. An application only needs to know about the interface, while the underlying DBD drivers provided by databases will do the actual work. For a Perl - DB2 application, the driver is the **DBD::DB2** driver, which provides the details for accessing a DB2 database.

Since the Perl DBI Module defines the standard database access interface, you can easily port an application written in DBI for accessing one DBMS, such as Oracle, to accessing another DBMS, such as IBM DB2. Moreover, the Perl DBI Module uses an interface that is quite similar to the ODBC and JDBC interfaces, which makes it easy to port Perl applications written in DBI to ODBC and JDBC interface, or vice versa.

**Note:**

For more information about Perl DBI, refer to <http://dbi.perl.org/>

For the latest DBD::DB2 driver and related information, visit

<http://www.ibm.com/software/data/db2/perl/> and <http://search.cpan.org/~ibmtordb2/>

## 9.2 Setting up the environment

In order to access a DB2 database from a Perl application, you need to set up a Perl environment to work with DB2. The following 3 steps are required:

1. Install the Perl language environment

You need to install Perl 5.8 or later.

On Linux/UNIX/Mac OS X, Perl is normally shipped as a standard component of the OS installation.

On Windows, you can download the ActiveState's free Perl binary distribution ActivePerl from <http://www.activestate.com/activeperl/>. ActivePerl is also available for free on Linux/UNIX. You can download the binaries from the website <http://www.activestate.com/activeperl/downloads/> and install Perl by running the Windows Installer on Windows or `install.sh` on Linux/UNIX.

Alternatively, you can download the source code of Perl interpreter from <http://dev.perl.org/> and then use appropriate C compilers to generate the Perl binaries for your platform. Refer to the above Web site for details about building and installing Perl from source.

After installation, you can check the version of Perl by running `perl -v`. If the version is lower than 5.8, you need to upgrade to 5.8 or later.

## 2. Install the Perl DBI module.

This is a prerequisite for any RDBMS including DB2 with its `DBD::DB2` driver module.

If you are using ActivePerl distribution and have internet access, you can install a binary version of the DBI module through ActivePerl's **Perl Package Manager** by issuing the following command:

```
ppm install DBI
```

The Perl Package Manager will download the DBI module and install it automatically.

Alternatively if you have internet access, you can run the following to build and install the DBI module from the source:

```
perl -MCPAN -e "install DBI"
```

The above command will download the DBI source code, then build and install it automatically through the **CPAN** module. Note that you need a C compiler for building the source. For details about the CPAN module, refer to <http://www.perl.com/doc/manual/html/lib/CPAN.html>.

If you don't have internet access, you can download the latest release of Perl DBI module source code from <http://search.cpan.org/~timb/DBI> at a later time, and then build the source and install the DBI module following the instructions shown in *Listing 9.1* for Linux/UNIX and *Listing 9.2* for Windows. You need a C compiler for building the source.

```
# the source is DBI-<x.xxx>.tar.gz, where x.xxx means version.
zcat DBI-<x.xxx>.tar.gz|tar xvf -
cd DBI-<x.xxx>
perl Makefile.PL
make
```

```
make test
make install
```

**Listing 9.1 – Install Perl DBI module from source on Linux/UNIX**

```
#unpack/unzip the source file DBI-<x.xxx>.tar.gz using winzip
cd DBI-<x.xxx>
perl Makefile.PL
nmake
nmake test
nmake install
```

**Listing 9.2 – Install Perl DBI module from source on Windows****Note:**

For more information about the Perl DBI API, visit <http://search.cpan.org/~timb/DBI/DBI.pm>

### 3. Install an IBM DB2 data server client

After installing all the above software, you will be able to install the Perl DB2 driver DBD::DB2. This is discussed in the next sections.

#### 9.2.1 Perl adapters and drivers

At the time of writing, the latest release of the DBD::DB2 driver module is 1.78. For more information about the DBD::DB2 driver, visit <http://www.ibm.com/db2/perl/> and <http://search.cpan.org/~ibmtordb2/>.

If you are using ActivePerl distribution on Windows and have the internet connection, you can install the DBD::DB2 module through ActivePerl's Perl Package Manager by issuing the following command:

For Perl version 5.8:

```
ppm install http://theoryx5.uwinnipeg.ca/ppms/DBD-DB2.ppd
```

For Perl Version 5.10:

```
ppm install http://cpan.uwinnipeg.ca/PPMPackages/10xx/DBD-DB2.ppd
```

Alternatively you can build and install the driver from source. You will need a C compiler. Download the latest release of DBD::DB2 module source from <http://www.cpan.org/authors/id/I/IB/IBMTORDB2> and follow the instructions in *Listing 9.3* (for Linux/UNIX) and *Listing 9.4* (for Windows).

```
# the source is DBD-DB2-<x.xx>.tar.gz, where x.xx means version.
```

```

zcat DBD-DB2-<x.xx>.tar.gz|tar xvf -
cd DBD-DB2-<x.xx>
export DB2_HOME=/home/db2inst1/sqllib #for example
perl Makefile.PL
make
make test
make install

```

**Listing 9.3 – Install the DBD::DB2 driver from source in Linux/UNIX**

```

#unpack/unzip the source file DBD-DB2-<x.xx>.tar.gz using winzip
cd DBD-DB2-<x.xx>
perl Makefile.PL
nmake
nmake test
nmake install

```

**Listing 9.4 – Install the DBD::DB2 driver from source in Windows**

## 9.3 Developing Perl DB2 applications

In this section you will learn how to develop a Perl application that will access DB2 data by calling the Perl DBI API.

### 9.3.1 Connecting to a DB2 database

First, enable Perl to load the Perl DBI module that provides the standard DBI APIs used to access a database. This can be done using this line in your application:

```
use DBI;
```

Then, connect to the DB2 database by calling the `DBI->connect` method of the DBI package using this syntax:

```
$dbhhandle = DBI->connect($data_source, $userID, $password, \%attr);
```

*Table 9.1* explains some of the parameters of the `DBI->connect` method.

Name	Description
<code>\$data_source</code>	A database connection string with the format of <code>'dbi:DB2:dbalias'</code> , where <i>dbalias</i> represents one of the following: <ol style="list-style-type: none"> <li>1) A DB2 Database alias cataloged in your DB2 client, or</li> <li>2) A connection string that includes the database name, host name, port number, protocol, user ID, and password in the format of <code>"DATABASE=database; HOSTNAME=hostname; PORT=port; PROTOCOL=TCPIP; UID=username; PWD=password;"</code>. This format</li> </ol>

	can be used for connecting to a remote database via TCPIP directly without first cataloging it on the client
<code>\$userID</code>	The user ID used to connect to the database, it's optional if <i>username</i> is specified in <code>\$data_source</code> connection string
<code>\$password</code>	The password for the user ID to connect to the database. it's optional if <i>password</i> is specified in <code>\$data_source</code> connection string
<code>\%attr</code>	Optional, a reference to the list of database connection attributes

**Table 9.1 - Parameters for the DBI->connect method**

The `DBI->connect` method will automatically load the `DBD::DB2` module if it was not loaded earlier, and return a database handle if the connection succeeds. The database handle will be used for all future function calls made on the DB connection.

For example, to connect to the *SAMPLE* database, make the connection as shown in *Listing 9.5*:

```
Use DBI;
my $dbhhandle = DBI->connect("dbi:DB2:sample", "db2admin", "password",
{AutoCommit=>0});
```

**Listing 9.5 - Connect to SAMPLE database cataloged on the client**

In the above listing, the database user ID is `db2admin`, the password is `password` and the `AutoCommit` connection attribute is set off, which means it is turned off. This means you can explicitly issue a commit or rollback for a transaction for this connection. By default `AutoCommit` is on. After successful execution, the `DBI->connect` returns a database handle in the variable `$dbhhandle`.

If the *SAMPLE* database is a remote database on a machine *host2* where the instance is listening to port 50000 and it has not been cataloged in your client, you can make a connection to it via TCP/IP directly as shown in *Listing 9.6*

```
use DBI;
my $dbhhandle = DBI->connect("dbi:DB2:DATABASE=sample; HOSTNAME=host2;
PORT=50000; PROTOCOL=TCPIP;UID=db2admin;PWD=password", {AutoCommit=>0});
```

**Listing 9.6 - Connect to the remote SAMPLE database via TCP/IP directly**

### 9.3.2 Retrieving data

After connecting to the database, you can issue `SELECT` SQL statements to retrieve data from the database.

If the `SELECT` statement is not known at the time the application is written, refer to *section 9.3.4* on how to run a SQL that contains variable inputs or parameter markers.

If the `SELECT` statement is known, follow the steps below:

1. Connect to the database by calling the `DBI->connect` method.

2. Prepare a **SELECT** SQL statement by calling the database handle's **prepare** method, which returns a statement handle if the SQL is successfully prepared.

For example, you can call the **prepare** method with an **SELECT** statement passed in as a string argument as follows:

```
$sthandle = $dbh->prepare(
    "SELECT firstnme, lastname
    FROM employee WHERE workdept='A00' " );
```

Where:

**\$dbh** is the database handle returned from step 1 and **\$dbh->prepare** returns a statement handle in the variable **\$sthandle** if the statement is successfully prepared.

3. Execute the prepared **SELECT** statement by calling the **execute** method using the statement handle. After a successful execution of the SQL, the statement handle will be associated with the result set.

For example, you can execute the statement prepared in the previous step by using the following Perl statement:

```
$rc = $sthandle->execute();
```

4. Fetch rows from the result set associated with the statement handle by calling **fetchrow\_array()** method on the statement handle, which returns a row as an array with one value for each column.

For example, you can fetch the rows from the above statement handle **\$sthandle** as shown in *Listing 9.7*:

```
while (($firstnme, $lastname) = $sthandle->fetchrow_array()) {
    print "$firstnme $lastname\n";
}
```

**Listing 9.7 – Fetch rows from the result set**

**Note:**

In addition to **fetchrow\_array**, the statement handle also provides many other data fetching methods, such as **fetchrow\_arrayref**, **fetchrow\_hashref**, **fetchall\_arrayref**, **fetchall\_hashref**, to support flexible ways of fetching rows from a result set. For more information about these functions, refer to <http://search.cpan.org/~timb/DBI/DBI.pm>.

The above code snippets are part of the script `select.pl` included in the **Exercise\_Files\_DB2\_Application\_Development.zip** file accompanying the book.

You can test the code by modifying the user ID and password in the script appropriately and run it as:

```
perl select.pl
```

The result of executing this script is illustrated as in *Figure 9.2*:

```
DB2 CLP
C:\books\db2app\samples\Chapter22>perl select.pl
The query will return 2 fields.

Field names:FIRSTNAME LASTNAME
CHRISTINE HAAS
UINCENZO LUCCHESI
SEAN O'CONNELL
DIAN HEMMINGER
GREG ORLANDO

C:\books\db2app\samples\Chapter22>
```

**Figure 9.2 - Executing select.pl**

### 9.3.3 Inserting, updating, and deleting data

To issue **INSERT/UPDATE/DELETE** SQL statements that are known at the time the application is written, you can call the database handle's `do()` method. Otherwise, to run SQL statements that contain variable inputs or parameter markers refer to *section 9.3.4*.

The syntax for the `do()` method is as follows:

```
$cnt = $dbh->do($statement);
```

*Table 9.2* describe the parameters of the `do()` method in more detail.

Name	Description
<code>\$statement</code>	A statement string
<code>\$cnt</code>	The number of rows affected by the SQL statement or <code>undef</code> on error. A return value of -1 means the number of rows is not known, not applicable, or not available.

**Table 9.2 - Parameters for the do() method of the database handle**

*Listing 9.8* provides several examples that illustrate how to use the `do` method to issue **INSERT/UPDATE/DELETE** statements and also run **CREATE/DROP TABLE** DDLs.

```
(1) $cnt = $dbhandle->do ("CREATE TABLE product(product_id CHAR(6),
                        name CHAR(30))");
(2) $cnt = $dbhandle->do ("INSERT INTO product VALUES
                        ('000001','computer'),('000002','TV')")
```



```

(3) print "Returns for insert: $cnt \n";
(4) $cnt = $dbh->do ("UPDATE product SET name = 'notebook'
                    WHERE product_id='000001'");
(5) print "Returns for update: $cnt \n";
(6) $cnt = $dbh->do ("DELETE FROM product WHERE
                    product_id='000003'");
(7) print "Returns for delete: $cnt \n";
(8) $cnt = $dbh->do ("DROP TABLE product");

```

#### Listing 9.8 – execute INSERT/UPDATE/DELETE statements

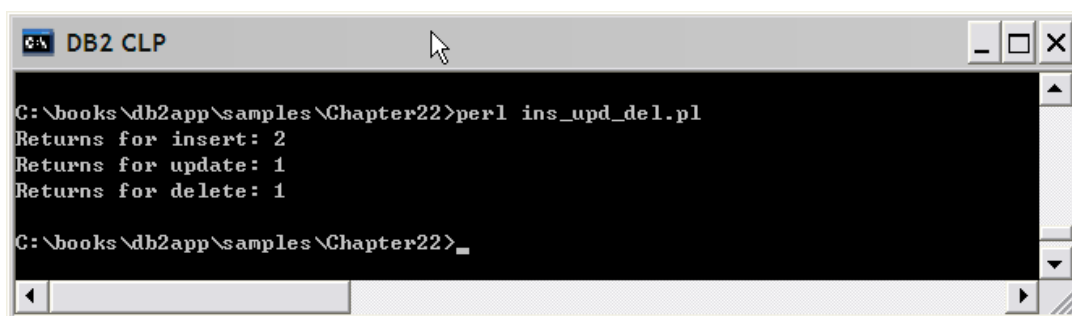
Items in *Listing 9.8* are explained as follows:

- (1) Call the method `$dbh->do` to create the table *product*
- (2) Call the method `$dbh->do` to insert rows into the *product* table
- (3) Print the number of rows inserted
- (4) Call the method `$dbh->do` to update some rows in the table
- (5) Print the number of rows updated
- (6) Delete some rows from the table
- (7) Print the number of rows deleted
- (8) Drop the table.

The above code snippets are part of the script `ins_upd_del.pl` included in the exercise files accompanying the book. You can test the code by modifying the user ID and password in the script appropriately and run it as follows:

```
perl ins_upd_del.pl
```

The result of executing the script `ins_upd_del.pl` is illustrated as in *Figure 9.3*:



```

C:\books\db2app\samples\Chapter22>perl ins_upd_del.pl
Returns for insert: 2
Returns for update: 1
Returns for delete: 1

C:\books\db2app\samples\Chapter22>_

```

Figure 9.3 Executing `ins_upd_del.pl`

### 9.3.4 Executing a SQL statement with parameter markers

In this section, you will learn how to issue an SQL statement that includes parameter markers or variable inputs. A parameter marker in an SQL statement, is represented by the question mark (?) character or a colon followed by a name (:*name*).

To run a SQL statement with parameter markers, you can follow these steps:

1. Connect to the database by calling the `DBI->connect` method
2. Prepare the statement by calling the `prepare` method on the database handle, which returns a statement handle if prepare is successful. For example, you can prepare a `SELECT` statement containing a parameter marker as follows:

```
$sthandle = $dbh->prepare( 'SELECT empno, lastname, job, salary
                           FROM employee WHERE workdept = ?' );
```

The `SELECT` statement above contains a parameter marker "?" in the predicate "workdept = ?", where the parameter marker "?" represents a variable input for a department number that is not known until the SQL is run. If the SQL statement is successfully prepared, the `prepare` method returns a statement handle and assigns it to the variable `$sthandle`.

3. Call `bind_param` method on the statement handle to bind parameters to local Perl variables.

The syntax of `bind_param` method is:

```
$sth->bind_param($p_num, $bind_value, \%attr)
```

Table 9.3 provides a description of the parameters of `bind_param`.

Name	Description
<code>\$p_num</code>	Specifies the 1-indexed position of the parameter in the SQL containing the parameter markers.
<code>\$bind_value</code>	A Perl variable or value to be bound to the parameter specified by parameter-number
<code>\%attr</code>	Optional: it can be used to indicate the parameter's type, precision, scale.

**Table 9.3 - Parameters for the `bind_param` method of the statement handle**

For example, you can call `bind_param` to bind the value "A00" to the parameter in the prepared `SELECT` statement shown earlier as follows:

```
$sthandle->bind_param(1, "A00");
```

After a successful execution, the value "A00" will be bound to the parameter marker (?) in the SQL's predicate "workdept = ?".

4. Call `execute` method on the statement handle to execute this statement. If it's a `SELECT` statement or a `CALL` to a stored procedure returning result sets (calling stored procedures will be discussed further in later sections), the statement handle will be associated with the result set after a successful call to the `execute` method. You can then use fetch functions, such as `fetchrow_array()`, to retrieve rows from the result set as discussed in the previous *section 9.3.2*.

For example, you can execute the above `SELECT` statement as follows:

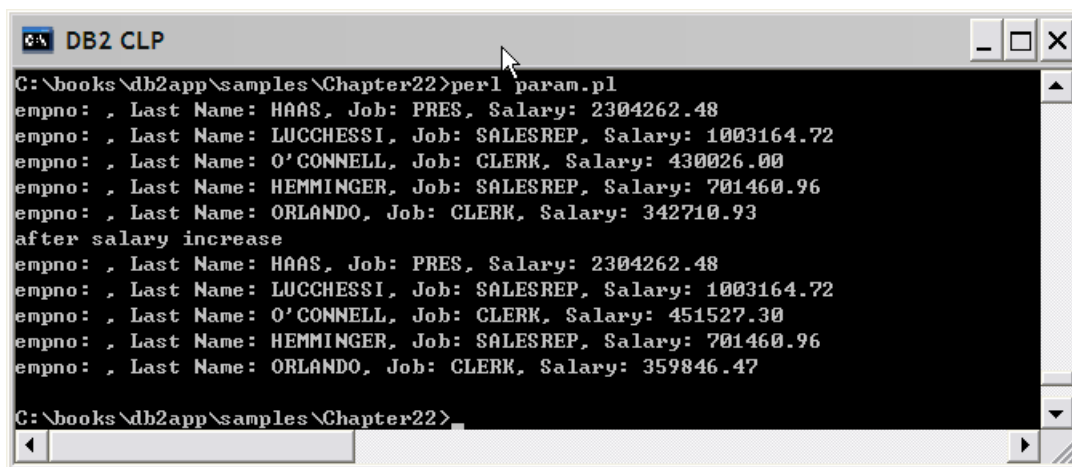
```
$sth->execute();
```

After successful execution of the SQL statement, a result set is associated with the statement, you can then begin fetching rows from the result set.

The above code snippets are part of the script `param.pl` included as part of the exercise files accompanying the book. In addition to `SELECT` statement, the script also contains an example of executing an `UPDATE` statement with parameter markers. If you want to test the code snippets, you can modify the user ID and password in the script appropriately and run it as:

```
perl param.pl
```

The result of executing the script `param.pl` is illustrated in *Figure 9.4*:



```

C:\books\db2app\samples\Chapter22>perl param.pl
empno: , Last Name: HAAS, Job: PRES, Salary: 2304262.48
empno: , Last Name: LUCCHESSI, Job: SALESREP, Salary: 1003164.72
empno: , Last Name: O'CONNELL, Job: CLERK, Salary: 430026.00
empno: , Last Name: HEMMINGER, Job: SALESREP, Salary: 701460.96
empno: , Last Name: ORLANDO, Job: CLERK, Salary: 342710.93
after salary increase
empno: , Last Name: HAAS, Job: PRES, Salary: 2304262.48
empno: , Last Name: LUCCHESSI, Job: SALESREP, Salary: 1003164.72
empno: , Last Name: O'CONNELL, Job: CLERK, Salary: 451527.30
empno: , Last Name: HEMMINGER, Job: SALESREP, Salary: 701460.96
empno: , Last Name: ORLANDO, Job: CLERK, Salary: 359846.47
C:\books\db2app\samples\Chapter22>

```

Figure 9.4 - Executing `param.pl`

### 9.3.5 Calling a stored procedure

In order to call a DB2 stored procedure from a Perl application, you need to perform the following steps:

1. Connect to the database by calling the `DBI->connect` method.
2. Prepare the `CALL` stored procedure statement by calling the `prepare` method on the database handle, which returns a statement handle if prepare is successful.

For example, let's say you want to call the SQL procedure *sp\_get\_employees* shown in *Listing 9.9*.

```
CREATE PROCEDURE sp_get_employees (IN dept_no CHAR(3), OUT dept_name
VARCHAR(36))
DYNAMIC RESULT SETS 1
BEGIN
    DECLARE emp_cursor CURSOR WITH RETURN TO CLIENT FOR
        SELECT firstname, lastname FROM employee WHERE workdept=dept_no;
    OPEN emp_cursor;
    SELECT deptname INTO dept_name FROM department WHERE deptno=dept_no;
END @
```

#### Listing 9.9 – sp\_get\_employees.db2

*sp\_get\_employees* has two parameters, the input parameter *dept\_no* and the output parameter *dept\_name*. It also returns one result set to the client which is defined in the cursor *emp\_cursor*.

You can prepare the **CALL** statement as follows:

```
$sthandle = $dbh->prepare( "CALL sp_get_employees(?,?)" );
```

The two parameters are represented as the parameter markers "?" in the **CALL** statement. After a successful prepare, the statement handle is returned in the variable *\$sthandle*.

3. Call **bind\_param** or **bind\_param\_inout** method on the statement handle to bind parameters to local Perl variables or values. **bind\_param** can be used for binding **IN** parameters only, while **bind\_param\_inout** is used for binding **IN/INOUT/OUT** parameters.

Refer to the previous section for the syntax of **bind\_param**. The syntax of **bind\_param\_inout** method is:

```
$rv = $sth->bind_param_inout($p_num, \$bind_value, $max_len, \%attr)
```

*Table 9.4* provides details about the parameters of **bind\_param\_inout**

Name	Description
<i>\$p_num</i>	Specifies the 1-indexed position of the parameter in the <b>CALL</b> statement.
<i>\$bind_value</i>	A Perl variable or value to be bound to the parameter specified by <i>\$p_num</i> . If it's <b>OUT/INOUT</b> parameter, a variable is used, otherwise it can be a variable or a value.
<i>\$max_len</i>	Specifies the minimum amount of memory to allocate to <i>\$bind_value</i> .

<code>\%attr</code>	Optional: it can be used to indicate the parameter's type, precision, scale.
---------------------	------------------------------------------------------------------------------

**Table 9.4 - Parameters for the `bind_param_inout` method of the statement handle**

For example, given the prepared `CALL` statement in step 2, you can call `bind_param` to bind the value "A00" to the first input parameter of the procedure and call `bind_param_inout` to bind the variable `$deptname` to the second output parameter as in *Listing 9.10* below:

```
$sthandle->bind_param(1, "A00");
$sthandle->bind_param_inout (2, \$deptname, 36,
{ 'TYPE' => SQL_VARCHAR });
```

**Listing 9.10 – bind the parameters**

4. Call the `execute` method on the statement handle to execute this `CALL` statement. If the SQL procedure is to return one or more result sets, a successful call to `execute` will associate the result sets with the statement handle. You can then use fetch functions, such as `fetchrow_array()` to retrieve rows from the result set as discussed in the previous *section 9.3.2*. If there are multiple result sets, you can use `db2_more_results` method of the statement handle, for example `$sth->{db2_more_results}`, to move to the next result set.

To execute the `CALL` statement prepared in previous steps, issue:

```
$sthandle->execute();
```

After successful execution of the `CALL` statement, a result set is associated with the statement. You can then begin fetching rows from the result set.

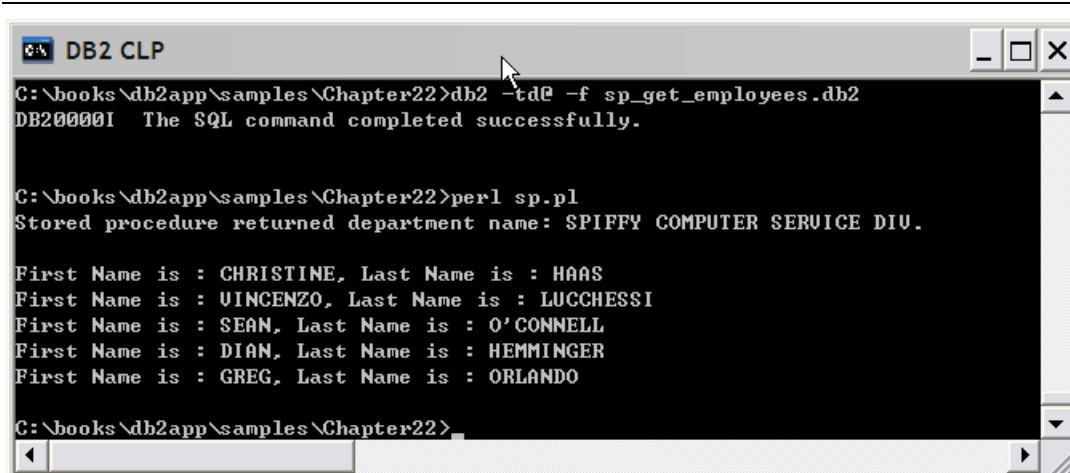
The above code snippets are part of the script `sp.pl` included with the exercise files accompanying this book. If you want to test the code snippets, you need to run the following commands to first create the SQL procedure `sp_get_lastname` as defined in the db2 script `sp_get_employees.db2` accompanying the book:

```
db2 connect to sample
db2 -td@ -f sp_get_employees.db2
db2 terminate
```

You should also modify the user ID and password in the `sp.pl` script appropriately and then run it as:

```
perl sp.pl
```

The result of executing the script `sp.pl` is illustrated in *Figure 9.5*.



```

C:\books\db2app\samples\Chapter22>db2 -td@ -f sp_get_employees.db2
DB20000I The SQL command completed successfully.

C:\books\db2app\samples\Chapter22>perl sp.pl
Stored procedure returned department name: SPIFFY COMPUTER SERVICE DIV.

First Name is : CHRISTINE, Last Name is : HAAS
First Name is : UINCENZO, Last Name is : LUCCHESSI
First Name is : SEAN, Last Name is : O'CONNELL
First Name is : DIAN, Last Name is : HEMMINGER
First Name is : GREG, Last Name is : ORLANDO

C:\books\db2app\samples\Chapter22>

```

Figure 9.5 - Executing sp.pl

## 9.4 Exercises

### Exercise #1

In this exercise, you will setup a Perl DB2 environment on a SUSE Linux Enterprise Server 10 SP1, assuming a valid DB2 V9.7 product has been installed first. Please note that you have to be the *root* user to perform the setup. The setup procedure is similar if you are using other operating systems.

1. Install Perl 5.8 or later.

Perl 5.8.8 comes with the standard installation of SUSE Linux Enterprise Server 10 SP1, so it's not required to install the Perl language environment separately in this exercise.

You can run the following command in the shell to check the version of your Perl language:

```
perl -v
```

If your version of Perl is lower than 5.8, follow *section 9.2* on how to install Perl language on your platform.

2. Install the Perl DBI module
  - Download the latest Perl DBI source file from <http://search.cpan.org/~timb/DBI/>

At the time of this writing, the latest version of DBI source is `DBI-1.611.tar.gz`

  - Login to the Linux server as the *root* user and perform the following commands in the directory where the source file `DBI-1.611.tar.gz` is located:

```
tar xvfz DBI-1.611.tar.gz
cd DBI-1.611
perl Makefile.PL
```

```
make
make test
make install
```

### 3. Install DBD::DB2 module

- Download the latest DBD::DB2 source file from <http://www.ibm.com/software/data/db2/perl/>

At the time of this writing, the latest version of DBD::DB2 source is `DBD-DB2-1.78.tar.gz`

- As *root*, perform the following commands:

```
export DB2_HOME=/home/db2inst1/sqllib
perl Makefile.PL
make
make test
make install
```

#### Note:

`/home/db2inst1` is the home directory of the DB2 instance owner user and `/home/db2inst1/sqllib` is where the instance binary files are located. This assumes the instance owner has been set to `db2inst1`.

## Exercise #2

In this exercise, you will practice writing a small script to access data from the **SAMPLE** database.

1. Log on to the server as the instance owner (for example `db2inst1` or `db2admin` on Windows).
2. If the **SAMPLE** database has not been created earlier, run the following command from a DB2 command window or Linux shell to create it.

```
db2samp1
```

3. Write a Perl script to print out all the employees who are working in the department of "SOFTWARE SUPPORT". At the same time increase the salary by 5% for each employee who was hired before '1996-01-01' in this department.
4. If you have problems creating this script, the solution is provided in the script `perl_ex1.pl` accompanying the book. You can modify the user ID and password in the script appropriately and test it out as follows:

```
perl perl_ex1.pl
```

## 9.5 Summary

In this chapter, you have learned how to set up a Perl environment to work with DB2, and how to execute SQL statements and call stored procedures from Perl applications by calling the standard Perl DBI interface.

## 9.6 Review questions

1. Which API can a Perl application use to access a DB2 database?
2. What's the relationship between the Perl DBI and the database DBD drivers?
3. Which of the following is the standard database interface for a Perl script to access a DB2 database?
  - A. CLI
  - B. Embedded Perl
  - C. Perl DBI
  - D. JDBC
  - E. None of the above
4. Which of the following are the steps you can use to execute a SQL with parameter marker?
  - A. prepare, bind\_param, execute
  - B. prepare, bind\_variable, do
  - C. prepare, bind\_param, do
  - D. prepare, do, fetchrow\_array
  - E. None of the above
5. Which of the following functions can be used to execute a SQL directly without first preparing it?
  - A. exec\_immediate
  - B. execute
  - C. do
  - D. execute\_imm
  - E. None of the above



# 10

## Chapter 10 –Application development with Python

**Python** is a platform independent interpreted programming language which is free and available on many operating systems including Windows, Linux, UNIX and Mac OS X.

Python is an open source language environment managed by the **Python Software Foundation** (<http://www.python.org/>). Python's object-oriented features, and flexible data typing, together with its extensive standard libraries, makes it a popular language for rapid application development, including database applications.

In this chapter you will learn about:

- Setting up the Python environment to work with DB2
- Connecting to a DB2 database from a Python application
- Developing Python applications to access DB2 data

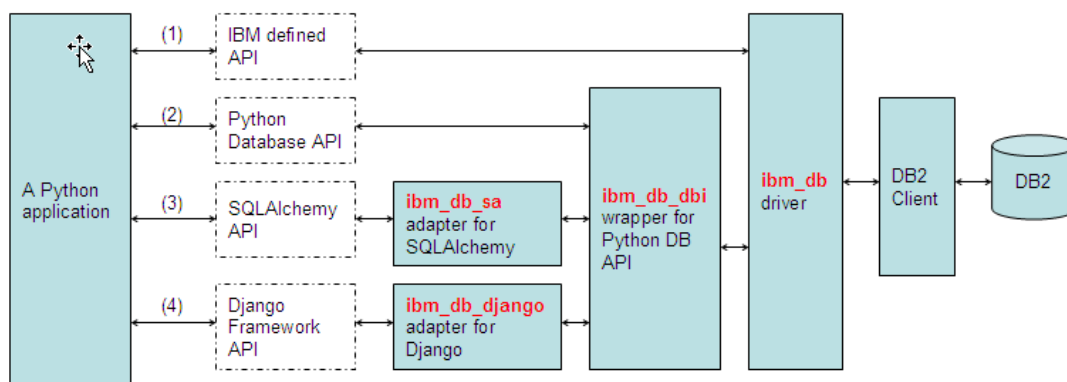
### Note:

If you are new to the Python programming language, review the free eBook [Getting Started with Python](#) which is part of this book series.

This chapter assumes you have created the SAMPLE database included with DB2. If you haven't, create it using the command `db2samp1`.

### 10.1 Python - DB2 applications: The big picture

Python applications access DB2 data servers through Python DB2 APIs and their drivers/adapters. Depending on the database API you want to work with, there are different types of drivers/adapters you can choose from. The high level view of these APIs and their drivers/adapters is illustrated in the *Figure 10.1*.



**Figure 10.1- High level view of Python DB2 APIs and drivers/adapters to access DB2**

As shown in *Figure 10.1*, there are four different types of database APIs and their drivers/adapters you can choose in a Python application to access a DB2 data server. The APIs and their drivers/adapters are explained in the next sections.

### 10.1.1 IBM defined API and `ibm_db` driver

This is a set of proprietary Python database APIs defined by IBM. The `ibm_db` driver is an implementation of the API. Through this API, you can not only issue SQL statements, call stored procedures, and use *pureXML*, but also access DB2 metadata information in a Python application.

The `ibm_db` driver is implemented as a Python module C extension to DB2's native **CLI/ODBC** interface; therefore, it can provide maximum performance and most advanced features. In later sections, we discuss about this driver in more detail.

#### Note:

The `ibm_db` API specification can be found at <http://code.google.com/p/ibm-db/wiki/APIs>

### 10.1.2 Python Database API and `ibm_db_dbi` driver

**Python Database API** is a set of open standard database APIs defined for a Python application. The `ibm_db_dbi` driver is simply a Python coded wrapper built upon the `ibm_db` driver to provide database APIs conforming to the standard **Python Database API** Specification, and similarly to `ibm_db` driver it enables you to issue SQL statements and call stored procedures through these standard APIs. Because this API conforms to the standard specification, it does not offer some of the advanced features that the `ibm_db` API supports. However, if you have an application written with a driver that supports Python Database API Specification, you can easily switch to `ibm_db_dbi`.

At the time of this writing, Python Database API Specification v2.0 is supported by `ibm_db_dbi` driver.

#### Note:

The Python Database API is a standard specification for the implementation of a Python interface to a database management system. You can find details about Python Database API Specification at the <http://www.python.org/dev/peps/pep-0249/>.

### 10.1.3 SQLAlchemy and ibm\_db\_sa adapter

SQLAlchemy is an open source Python SQL toolkit and **Object Relational Mapper** that gives application developers the full power and flexibility of SQL. The core of SQLAlchemy is called "**SQL expression language**" which allows users to access a database via Python functions and expressions using function-based query construction, instead of using normal SQL statements. The **ibm\_db\_sa** adapter is a Python coded DB2 adapter built upon the **ibm\_db\_dbi** driver and provides the support for **SQLAlchemy** APIs.

At the time of writing, SQLAlchemy 0.4 specification APIs are supported by the **ibm\_db\_sa** adapter.

#### Note:

You can find more information about SQLAlchemy, including the API reference at <http://www.sqlalchemy.org/>

### 10.1.4 Django framework and ibm\_db\_django adapter

Django is a popular open source Python web framework for rapid application development. Similar to SQLAlchemy, Django framework APIs enables you to access database via Python functions and expressions rather than using normal SQL statements. The **ibm\_db\_django** adapter is a Python coded DB2 adapter built upon the **ibm\_db\_dbi** driver and provides the support for the **Django** framework APIs.

At the time of this writing, Django 1.2 specification APIs are supported.

#### Note:

You can find more information about Django framework, including API reference at <http://www.djangoproject.com/>.

## 10.2 Setting up the environment

To set up the Python environment to work with DB2, follow these steps:

1. Install Python 2.5 or greater.

You can install Python using one of the following options:

- Install it using ActivePython binary distribution.

ActivePython is a free Python binary distribution provided by ActiveState Software and is available on most operating systems. You can download it from the website <http://www.activestate.com/activepython/downloads/> and install Python by running the Windows Installer on Windows or `install.sh` on Linux/UNIX.

- On Windows, you can alternatively download the free Python Windows Installer from <http://www.python.org/download/> and install it by running the Windows Installer.
- Of course, for all the operating systems, you can alternatively build from the source which can be freely downloaded from <http://www.python.org/download/>.

For example, to build and install Python 2.5.4 on Linux from the source, you can download the source `Python-2.5.4.tgz` and unpack it with "`tar -zxvf Python-2.5.4.tgz`", then change to the `Python-2.5.4` directory and run the "`./configure`", "`make`", "`make install`" commands to compile and install Python.

You can verify the Python version by running:

```
python -V
```

## 2. Install `setuptools`

`setuptools` is a free program to download, build, install, upgrade, and uninstall Python packages. It can be downloaded from <http://pypi.python.org/pypi/setuptools/> and you can install it following the instructions from this site. You may need it later to install some of the Python DB2 drivers.

## 3. Install an IBM DB2 data server client

After installing the above software, you will be able to install the Python DB2 adapters and drivers.

### 10.2.1 Python adapters & drivers

All Python-DB2 adapters and drivers can be freely downloaded from <http://code.google.com/p/ibm-db/downloads/list>. Alternatively the `ibm_db/ibm_db_dbi` drivers and source are also available at [http://pypi.python.org/pypi/ibm\\_db](http://pypi.python.org/pypi/ibm_db).

As discussed in previous section, you can decide which drivers or adapters to install based on the database APIs you want to use. For example, if you want to issue raw SQL statements through either the IBM defined API or Python Database API, you can just install the `ibm_db` and `ibm_db_dbi` drivers. If you want to use SQLAlchemy or the Django framework, you will have to install not only the `ibm_db_sa` or `ibm_db_django` adapter, but also the `ibm_db/ibm_db_dbi` driver which these adapters are built upon.

The install procedures for each driver or adapter are explained in the next sections.

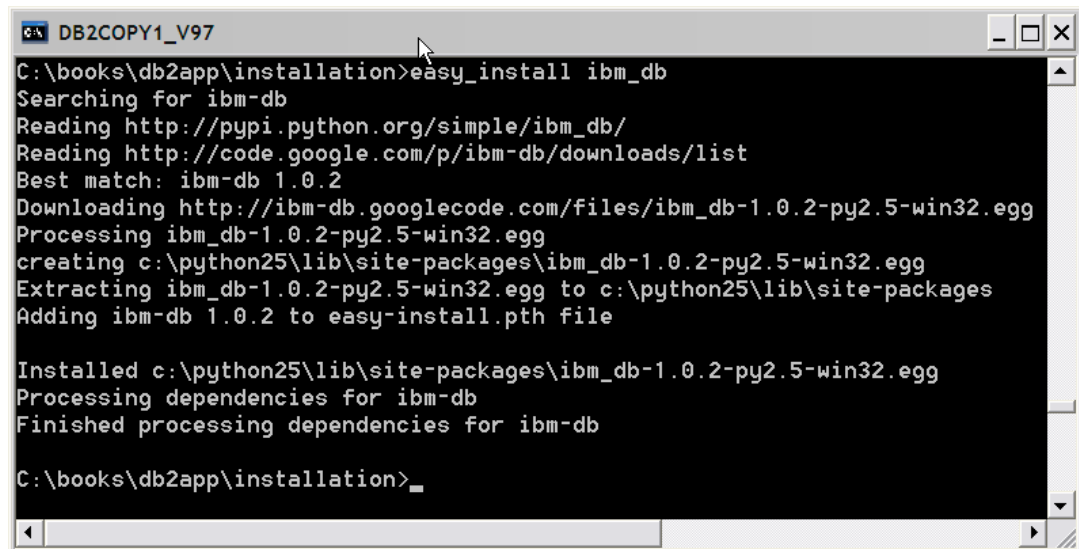
#### 10.2.1.1 Installing the `ibm_db` and `ibm_db_dbi` drivers

The `ibm_db` and `ibm_db_dbi` drivers are bundled in one installation package known as the `ibm_db` package. They should be installed together. At the time of writing, the latest release of these drivers is 1.0.2, and they are supported on Linux and Windows.

Follow this procedure to install and setup these two drivers:

1. Install the `ibm_db` package

If you have internet access, issue the following command as shown in *Figure 10.2* on Windows: `easy_install ibm_db`



```
DB2COPY1_V97
C:\books\db2app\installation>easy_install ibm_db
Searching for ibm-db
Reading http://pypi.python.org/simple/ibm_db/
Reading http://code.google.com/p/ibm-db/downloads/list
Best match: ibm-db 1.0.2
Downloading http://ibm-db.googlecode.com/files/ibm_db-1.0.2-py2.5-win32.egg
Processing ibm_db-1.0.2-py2.5-win32.egg
creating c:\python25\lib\site-packages\ibm_db-1.0.2-py2.5-win32.egg
Extracting ibm_db-1.0.2-py2.5-win32.egg to c:\python25\lib\site-packages
Adding ibm-db 1.0.2 to easy-install.pth file

Installed c:\python25\lib\site-packages\ibm_db-1.0.2-py2.5-win32.egg
Processing dependencies for ibm-db
Finished processing dependencies for ibm-db

C:\books\db2app\installation>
```

**Figure 10.2-** install `ibm_db/ibm_db_dbi` from the internet using `easy_install`

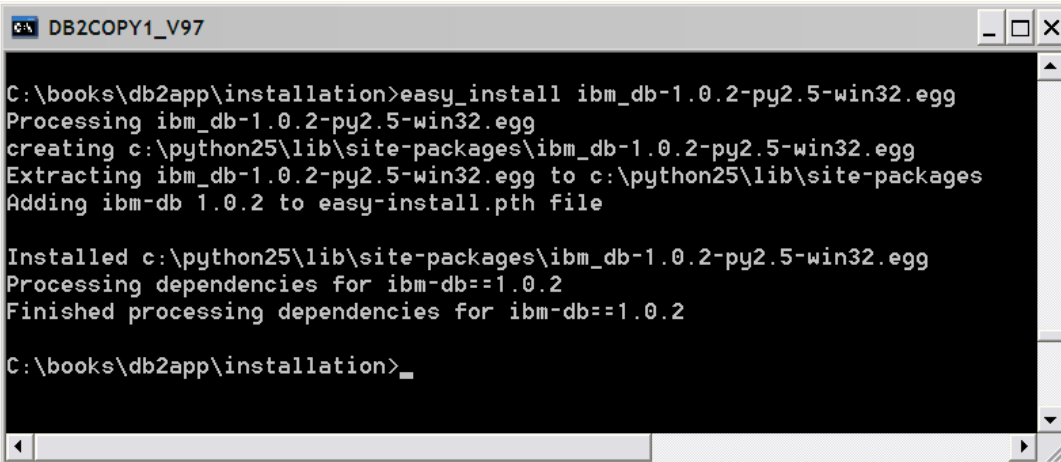
As shown in *Figure 10.2*, the `easy_install` command downloads the package from the internet and installs the two drivers under the `site-packages` directory where `setuptools` is installed. `easy_install` is a program provided by the `setuptools` package that was installed earlier.

If you do not have internet access, download the appropriate Python egg file for your platform from <http://code.google.com/p/ibm-db/downloads/list>, and issue the following command:

```
easy_install <egg_file_name>
```

where `<egg_file_name>` includes the path to the egg file.

For example, you can install the drivers using the downloaded egg file `ibm_db-1.0.2-py2.5-win32.egg` as shown in the *Figure 10.3*.



```

C:\books\db2app\installation>easy_install ibm_db-1.0.2-py2.5-win32.egg
Processing ibm_db-1.0.2-py2.5-win32.egg
creating c:\python25\lib\site-packages\ibm_db-1.0.2-py2.5-win32.egg
Extracting ibm_db-1.0.2-py2.5-win32.egg to c:\python25\lib\site-packages
Adding ibm-db 1.0.2 to easy-install.pth file

Installed c:\python25\lib\site-packages\ibm_db-1.0.2-py2.5-win32.egg
Processing dependencies for ibm-db==1.0.2
Finished processing dependencies for ibm-db==1.0.2

C:\books\db2app\installation>_

```

**Figure 10.3 - install `ibm_db/ibm_db_dbi` from an egg file using `easy_install`**

You can also build and install the driver from source code. Download the code from [http://pypi.python.org/pypi/ibm\\_db/](http://pypi.python.org/pypi/ibm_db/). The instructions are available in the README file shipped with the driver source code. You will need a C compiler on your platform in order to compile the C programs included in the source code.

2. Create an environment variable named `PYTHONPATH`, and specify the path to where the `ibm_db` egg is installed. For example:

- On Windows:

```
PYTHONPATH=<setuptools_install_path>\site-
packages\<ibm_db-xx.egg>
```

- On Linux (BASH shell):

```
export PYTHONPATH=<setuptools_install_path>/site-
packages/<ibm_db-xx.egg>
```

3. From a command prompt, test your setup by typing `python` to get into the Python interactive interpreter and entering code similar to *Listing 10.1* to test a connection.

```

(1) import ibm_db
(2) ibm_db_conn = ibm_db.connect('SAMPLE', 'db2admin', 'password')
(3) import ibm_db_dbi
(4) conn = ibm_db_dbi.Connection(ibm_db_conn)
(5) conn.tables('SYSCAT', '%')

```

**Listing 10.1 - Test the installation by connecting to a DB2 database**

In *Listing 10.1*:

- (1) Imports the `ibm_db` module
- (2) Makes a connection to the `SAMPLE` database using the function `ibm_db.connect`, where "db2admin" is the user id and "password" is the

password. You should replace these values appropriately. We discuss more about `ibm_db.connect` in later sections.

- (3) Imports the `ibm_db_dbi` module
- (4) Calls the function `ibm_db_dbi.Connection`
- (5) Lists all the tables with the schema `SYSCAT`

### 10.2.1.2 Installing the `ibm_db_sa` adapter

At the time of writing, the latest release 0.1.6 of `ibm_db_sa` adapter which supports SQLAlchemy 0.4 is available on Linux and Windows.

Follow this procedure to install and setup the `ibm_db_sa` adapter:

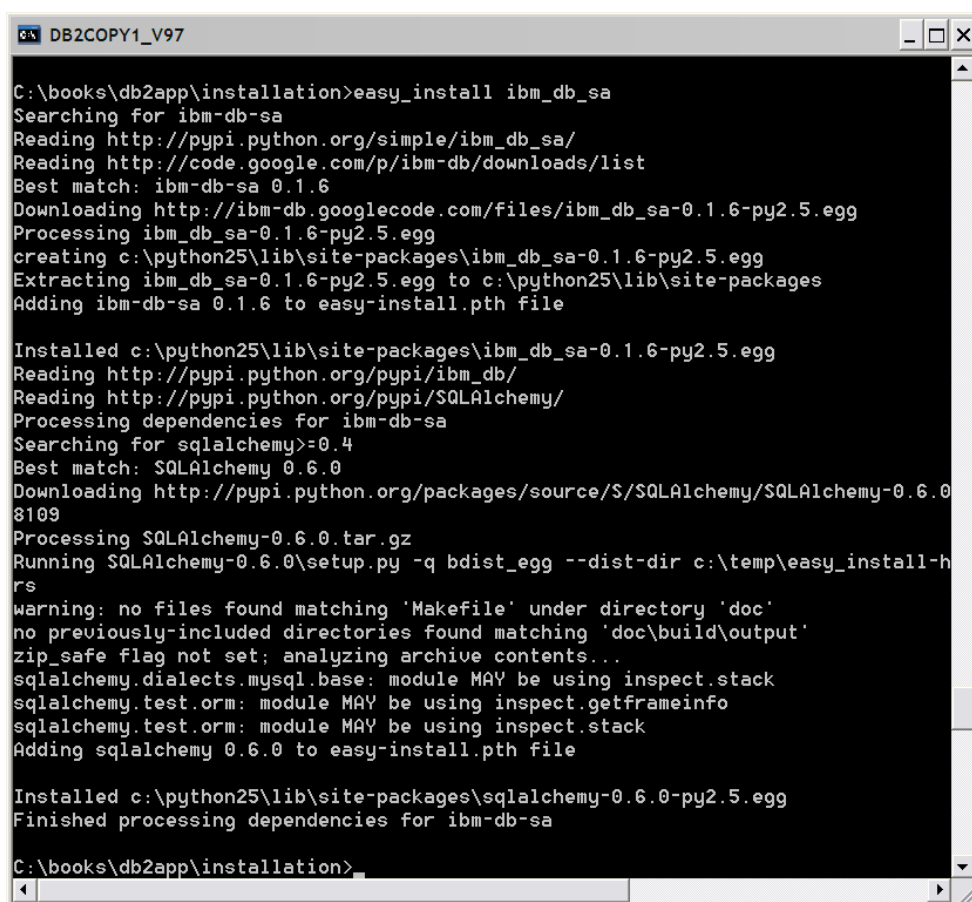
1. Install the `ibm_db_sa` package

If you have internet access, issue the following command:

```
easy_install ibm_db_sa
```

This command will download the package from the internet and install the driver under the `site-packages` directory. Note that since the adapter is dependent upon the `ibm_db/ibm_db_dbi` drivers and also the `SQLAlchemy` package, `easy_install` will download and install them if they have not been installed before.

For example, in Windows you can install the adapter from the internet as illustrated in *Figure 10.4*.



```

C:\books\db2app\installation>easy_install ibm_db_sa
Searching for ibm-db-sa
Reading http://pypi.python.org/simple/ibm_db_sa/
Reading http://code.google.com/p/ibm-db/downloads/list
Best match: ibm-db-sa 0.1.6
Downloading http://ibm-db.googlecode.com/files/ibm_db_sa-0.1.6-py2.5.egg
Processing ibm_db_sa-0.1.6-py2.5.egg
creating c:\python25\lib\site-packages\ibm_db_sa-0.1.6-py2.5.egg
Extracting ibm_db_sa-0.1.6-py2.5.egg to c:\python25\lib\site-packages
Adding ibm-db-sa 0.1.6 to easy-install.pth file

Installed c:\python25\lib\site-packages\ibm_db_sa-0.1.6-py2.5.egg
Reading http://pypi.python.org/pypi/ibm_db/
Reading http://pypi.python.org/pypi/SQLAlchemy/
Processing dependencies for ibm-db-sa
Searching for sqlalchemy>=0.4
Best match: SQLAlchemy 0.6.0
Downloading http://pypi.python.org/packages/source/S/SQLAlchemy/SQLAlchemy-0.6.0
8109
Processing SQLAlchemy-0.6.0.tar.gz
Running SQLAlchemy-0.6.0\setup.py -q bdist_egg --dist-dir c:\temp\easy_install-h
rs
warning: no files found matching 'Makefile' under directory 'doc'
no previously-included directories found matching 'doc\build\output'
zip_safe flag not set; analyzing archive contents...
sqlalchemy.dialects.mysql.base: module MAY be using inspect.stack
sqlalchemy.test.orm: module MAY be using inspect.getframeinfo
sqlalchemy.test.orm: module MAY be using inspect.stack
Adding sqlalchemy 0.6.0 to easy-install.pth file

Installed c:\python25\lib\site-packages\sqlalchemy-0.6.0-py2.5.egg
Finished processing dependencies for ibm-db-sa

C:\books\db2app\installation>

```

**Figure 10.4 - Install `ibm_db_sa` from the internet using `easy_install`**

As shown in *Figure 10.4*, the `easy_install` command downloads the `ibm_db_sa` package and all the prerequisite packages such as `ibm_db/ibm_db_dbi` and `SQLAlchemy` from the internet if they were not installed before and installs them under the site-packages directory where `setuptools` is installed.

If you do not have internet access, download the Python egg file from <http://code.google.com/p/ibm-db/downloads/list>, and issue the following command:

```
easy_install <egg_file_name>
```

where `<egg_file_name>` includes the path to the egg file.

For example:

```
easy_install ibm_db_sa-0.1.6-py2.5.egg
```

Note that you should first install the dependent `ibm-db` and `SQLAlchemy` package before you install `ibm_db_sa` package if you don't have internet access.



- From the command prompt, test your setup by typing **python** to launch the Python interpreter and entering code similar to *Listing 10.2* to test a DB2 connection.

```
(1) import sqlalchemy
    from sqlalchemy import *
(2) db2 = sqlalchemy.create_engine('ibm_db_sa://db2admin:password
@localhost:50000/SAMPLE')
(3) metadata = MetaData()
(4) users = Table('users', metadata,
    Column('user_id', Integer, primary_key = True),
    Column('user_name', String(16), nullable = False),
    Column('email_address', String(60), key='email'),
    Column('password', String(20), nullable = False)
    )
(5) metadata.bind = db2
(6) metadata.create_all()
```

#### Listing 10.2 - Test the installation by connecting to a DB2 database

In *Listing 10.2*:

- Imports the `sqlalchemy` package
- Creates a `db2` database Engine object, where "SAMPLE" is the **SAMPLE** database you are connecting to, "db2admin" is the user id, "password" is the password, and "localhost:50000" is the local instance listening at port 50000. You should replace them with your own values appropriately.
- Creates a `MetaData` object `metadata`
- Defines a table named `USERS`
- Binds the metadata object to `db2` engine
- Issues **CREATE** statements for all tables

#### 10.2.1.3 Installing `ibm_db_django` adapter

At the time of writing, the latest release 0.2.1 of the `ibm_db_django` adapter which supports Django 1.2, is available on Linux and Windows.

The procedure to install and setup the `ibm_db_django` adapter is as follows:

- Install the Django framework

Since the adapter `ibm_db_django` is dependent upon the Django framework package, you have to install the Django framework before installing the `ibm_db_django` adapter.

Install Django following the instructions from the Django Web site at

<http://docs.djangoproject.com/en/dev/topics/install/#installing-an-official-release>

If you are using Django 1.0.2, you also need to apply a patch in Django in order to remove non-standard SQL generation issue. For versions greater than 1.0.2 no patch is required. The details about the patch is located at <http://code.djangoproject.com/ticket/9862>

Extract `creation.py` file from the patch zip file at <http://code.djangoproject.com/changeset/9703?format=zip&new=9703>

Copy this `creation.py` to the Django installation directory at `site-packages/django/db/backends/`

2. Install the `ibm_db/ibm_db_dbi` drivers 0.7.2.5 or higher if they have not been installed. For details, refer to the previous section "*Installing the ibm\_db and ibm\_db\_dbi drivers*".

3. Install the `ibm_db_django` package

Download the `ibm_db_django` source code `ibm_db_django-x.x.x.tar.gz` from <http://code.google.com/p/ibm-db/downloads/list>, where `x.x.x` refers to the version of the driver.

Uncompress the `gz` file `ibm_db_django-x.x.x.tar.gz`

Issue the following command to install:

```
cd ibm_db_django
python setup.py install
```

4. Verify the installation of the `ibm_db_django` adapter by testing the connection to DB2

Create a new Django project by executing:

```
django-admin.py startproject myproj
```

Go to the newly created directory, and edit the `settings.py` file as illustrated in *Listing 10.3*.

```
(1) DATABASE_ENGINE = 'ibm_db_django'
(2) DATABASE_NAME = 'SAMPLE'
(3) DATABASE_USER = 'db2admin'
(4) DATABASE_PASSWORD = 'password'
```

**Listing 10.3 - settings.py**

where:

- (1) "`ibm_db_django`" means you want to access a DB2 database server. This is a new format starting with django adapter version 0.1.2. For earlier django adapter versions, use `DATABASE_ENGINE = 'db2'` instead.
- (2) "`SAMPLE`" refers to the **SAMPLE** database.

(3) "db2admin" is the user ID, you can replace it with your own.

(4) "password" is the password, you can replace it with your own.

Run the test suite as:

```
python manage.py test
```

**Note:**

The open source project home for all the above four Python DB2 drivers can be found at the <http://code.google.com/p/ibm-db/>.

### 10.3 Developing Python DB2 applications

For simplicity purposes, this section focuses only on the `ibm_db` driver, which provides the maximum performance and most advanced features support for accessing DB2 data servers. For all other drivers or adapters, refer to their API specifications.

#### 10.3.1 Connecting to a DB2 database

You must first include the following line in your Python script to import the `ibm_db` driver module:

```
import ibm_db
```

Then, connect to a DB2 database by calling the function `ibm_db.connect` with the following syntax:

```
IBM_DBConnection ibm_db.connect (string dsn, string user, string
password[, array options])
```

Table 10.1 explains the parameters of `ibm_db.connect`.

Name	Description
<code>dsn</code>	Database connection string in the format of "DATABASE=database; HOSTNAME=hostname; PORT=port; PROTOCOL=TCPIP; UID=username; PWD=password;"
<code>user</code>	The username used to connect to the database; specify an empty string "" if username is specified in the <code>dsn</code> string
<code>password</code>	The password for the username connecting to the database; specify an empty string "" if password is specified in the <code>dsn</code> string
<code>options</code>	Optional: A dictionary of connection options that affect the behavior of the connection.

**Table 10.1 - Parameters for function `ibm_db.connect`**

For example, if **SAMPLE** is a database alias cataloged locally in your client, you can make a connection to it as shown in *Listing 10.4*:

```
import ibm_db
conn = ibm_db.connect("SAMPLE","db2admin","password")
```

#### Listing 10.4 – Connect to the locally cataloged **SAMPLE** database

If the **SAMPLE** database is a remote database on a machine *host2* with the instance listening at port 50000 and it has not been catalogued in your client, you can make a connection to it via TCP/IP directly as shown in *Listing 10.5*

```
import ibm_db
conn = ibm_db.connect("DATABASE=SAMPLE; HOSTNAME=host2; PORT=50000;
PROTOCOL=TCPIP; UID=db2admin; PWD=password;", "", "")
```

#### Listing 10.5 – Connect to the remote **SAMPLE** database via TCP/IP directly

If the connection attempt is successful, the function `ibm_db.connect` returns an `ibm_db.IBM_DBConnection` object, which will be used in the future to perform further database operations specific to this connection, such as executing SQL statements to retrieve, insert, update or delete data.

#### Note:

The `ibm_db` driver also supports the creation of persistent connections, which remain active in the connection pool after being closed and allows subsequent connection request to reuse an existing connection if they have an identical set of credentials. For detailed information, check the API specification for function `ibm_db.pconnect` at <http://code.google.com/p/ibm-db/wiki/APIs>

### 10.3.2 Retrieving data

If the SQL statement is not known at the time the application is written, refer to *section 10.3.4* on how to run a SQL that contains variable inputs or parameter markers.

If the SQL statement is known, follow the steps below to select and fetch data:

1. Connect to the database with the `ibm_db.connect` function, which returns a database connection object if connection is successful.
2. Execute the SQL `SELECT` statement directly by calling the function `ibm_db.exec_immediate`, which returns a statement object that is associated with the result set if the SQL statement executes successfully.

The syntax of function `ibm_db.exec_immediate` is shown below:

```
IBM_DBStatement ibm_db.exec_immediate( IBM_DBConnection connection,
string statement [, array options] )
```

*Table 10.2* explains the different parameters of `ibm_db.exec_immediate`.

Name	Description
<code>connection</code>	A valid database connection object as returned from <code>ibm_db.connect()</code>
<code>statement</code>	An SQL statement in string format. The statement cannot contain any parameter markers, i.e the statement should be known at the time it is written.
<code>options</code>	Optional: A dictionary containing statement options. You can use this parameter to request a scrollable cursor for database servers that support this type of cursor. By default, a forward-only cursor is returned.

**Table 10.2 - Parameters for function `ibm_db.exec_immediate`**

For example, you can call `ibm_db.exec_immediate` function to run a `SELECT` statement passed in as a string argument as shown below:

```
stmt = ibm_db.exec_immediate(conn, 'SELECT firstnme, lastname FROM
employee fetch first 2 rows only')
```

Where the function takes the connection object `conn` returned in step 1 as the first function argument, and returns a statement object `stmt` which is associated with the query's result set if the execution is successful.

- Fetch rows from the result set by calling `ibm_db.fetch_tuple()`, which returns a tuple representing the next or requested row in the result set. Each column value is indexed by the 0-indexed column position in the tuple.

The syntax of function `ibm_db.fetch_tuple` is shown as follows:

```
array ibm_db.fetch_tuple(IBM_DBStatement stmt, [, int row_number])
```

**Table 10.3 explains the different parameters of `ibm_db.fetch_tuple`**

Name	Description
<code>stmt</code>	A valid statement object containing a result set.
<code>row_number</code>	Optional: 1-indexed row number for a specific row that you want to fetch from the result set. Passing this parameter results in a warning if the result set uses a forward-only cursor

**Table 10.3 - Parameters for function `ibm_db.fetch_tuple`**

The function returns `False` if there are no more rows left in the result set, or if the row requested by `row_number` does not exist in the result set.

For example, you can fetch and print all the rows from the result set as shown in *Listing 10.6*.

```
(1) mytuple = ibm_db.fetch_tuple(stmt)
(2) while mytuple != False:
```

```
(3)     print "First Name: %s, Last Name: %s" % (mytuple[0], mytuple[1])
(4) mytuple = ibm_db.fetch_tuple(stmt)
```

#### Listing 10.6 – Fetch rows from the result set

Items in *Listing 10.6* are explained as follows:

- (1) The function `ibm_db.fetch_tuple` returns each row from the result set as a tuple and assigns it to the variable `mytuple`;
- (2) Loop until the function returns `False` if there are no rows left in the result set;
- (3) Print each columns' value from the returned row by using the 0-indexed column position, such as `mytuple[0]`, `mytuple[1]`;
- (4) Fetch the next row into the variable `mytuple`.

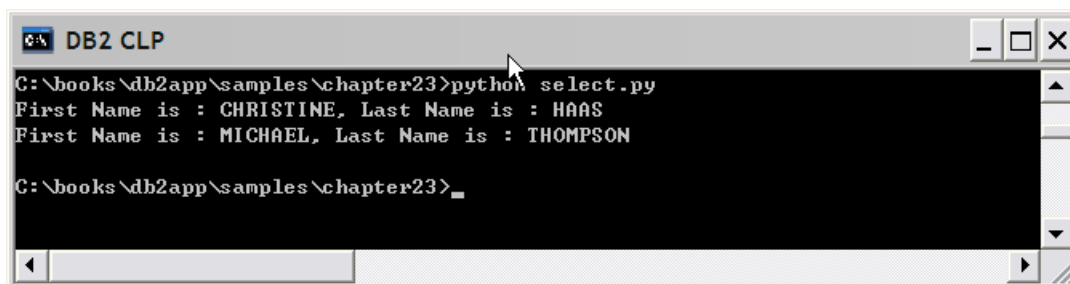
#### Note:

In addition to `ibm_db.fetch_tuple`, the `ibm_db` driver also provides 3 other data fetching functions that support flexible ways of fetching result sets: `ibm_db.fetch_assoc`, `ibm_db.fetch_both` and `ibm_db.fetch_row`. For more information review <http://code.google.com/p/ibm-db/wiki/APIs>.

If you would like to test all the above code snippets, edit the Python script `select.py` included in the exercise files accompanying this book and replace the user ID and password with your own. Then run the script as follows:

```
python select.py
```

The result of executing the above script is illustrated in *Figure 10.5*.



```
DB2 CLP
C:\books\db2app\samples\chapter23>python select.py
First Name is : CHRISTINE, Last Name is : HAAS
First Name is : MICHAEL, Last Name is : THOMPSON
C:\books\db2app\samples\chapter23>_
```

Figure 10.5 Executing the script `select.py`

#### Note:

All the scripts used in this chapter are included in the directory `samples/chapter10` in the zip file `Exercise_Files_DB2_Application_Development.zip` that accompanies this book.

To run the scripts, you may need to first create the `SAMPLE` database using the `db2samp1` utility. Also replace the user name and password in the scripts with your own ones.

### 10.3.3 Inserting, updating and deleting data

Similar to the `SELECT` statement, an `INSERT`, `UPDATE` or `DELETE` SQL statement that is already known at the time the application is written can be executed directly by calling the function `ibm_db.exec_immediate`. For a SQL statement that contains variable inputs or parameter markers, refer to *section 10.3.4*.

After successful execution of the SQL statements, you can use the `ibm_db.num_rows` function to return the number of rows that the SQL statement affected.

Let's take a look at the example in *Listing 10.7*.

```
(1) stmt = ibm_db.exec_immediate(conn, "CREATE TABLE PRODUCT(product_id
char(6), name char(30))")
(2) stmt = ibm_db.exec_immediate(conn, "Insert into PRODUCT values
('000001','computer'), ('000002','TV')")
(3) print "Returns for insert: ", ibm_db.num_rows(stmt)
(4) stmt = ibm_db.exec_immediate (conn, "update PRODUCT set name =
'notebook' where product_id='000001'")
(5) print "Returns for update: ", ibm_db.num_rows(stmt)
(6) stmt = ibm_db.exec_immediate (conn, "delete from PRODUCT where
product_id='000003'")
(7) print "Returns for delete: ", ibm_db.num_rows(stmt)
(8) stmt = ibm_db.exec_immediate (conn, "drop table PRODUCT")
```

#### Listing 10.7 - the script `ins_upd_del.py`

Items in *Listing 10.7* are explained as follows:

- (1) Calls the function `ibm_db.exec_immediate` to create the table `PRODUCT`
- (2) Calls the function `ibm_db.exec_immediate` to insert rows into the `PRODUCT` table
- (3) Prints the number of rows that were inserted
- (4) Updates some rows in the table
- (5) Prints the number of rows that were updated
- (6) Deletes some rows in the table
- (7) Prints the number of rows deleted
- (8) Drops the table.

The above code snippets are part of the script `ins_upd_del.py` included in the exercise files accompanying the book. You can test the code by modifying the user ID and password in the script appropriately and run it as:

```
python ins_upd_del.py
```

The result of executing the script is illustrated in *Figure 10.6*.

```

DB2 CLP
C:\books\db2app\samples\chapter23>python ins_upd_del.py
Returns for insert: 2
Returns for update: 1
Returns for delete: 0
C:\books\db2app\samples\chapter23>

```

Figure 10.6 - Executing the script `ins_upd_del.py`

### 10.3.4 Execute a SQL statement with parameter markers

In previous two sections, you have learned how to use function `ibm_db.exec_immediate` to execute SQL statements which is known at application written time. Now let's examine how to run a SQL statement which might contain variable inputs whose values are not known until at the run time.

The question mark (?) character will be used as the parameter marker for each variable input in the SQL statement string. The procedure to prepare and execute such SQL statements is explained below:

1. Connect to the database with the `ibm_db.connect` function, which returns a database connection object if connection is successful.
2. Prepare the SQL string with the parameter markers using the `ibm_db.prepare` function, which returns a `IBM_DBStatement` statement object if the prepare attempt is successful.

The syntax of function `ibm_db.prepare` is:

```

IBM_DBStatement ibm_db.prepare( IBM_DBConnection connection, string
statement [, array options] )

```

Table 10.4 describes the parameters of the function `ibm_db.prepare`

Name	Description
<code>connection</code>	A valid database connection object returned from <code>ibm_db.connect()</code>
<code>statement</code>	An SQL statement in string format, which may optionally contain input/output parameter markers, whose values are not known until the SQL is run.
<code>options</code>	Optional: A dictionary containing statement options. You can use this parameter to request a scrollable cursor for database servers that support this type of cursor. By default, a forward-only cursor is returned.

Table 10.4 - Parameters of the function `ibm_db.prepare`



The function returns a statement object if the SQL is successfully prepared, otherwise it raises an exception.

For example, you can prepare a **SELECT** statement containing a parameter marker as follows:

```
stmt = ibm_db.prepare(conn, "SELECT empno, lastname, job, salary FROM
employee WHERE workdept = ?")
```

As shown above, the connection object *conn* is passed in as the first function argument, while the second argument is a **SELECT** string which contains a parameter marker in the predicate "workdept = ?", where the parameter marker "?" represents a variable input for a department number that is not known until the SQL is run. If the function **ibm\_db.prepare** successfully prepares the SQL statement, it returns a statement object and assigns it to the variable *stmt*.

3. Bind input values to parameter markers in the SQL by calling the **ibm\_db.bind\_param** function for each parameter marker.

The syntax of function **ibm\_db.bind\_param** is:

```
Py_True/Py_None ibm_db.bind_param(IBM_DBStatement stmt, int
parameter-number, string variable [, int parameter-type [, int data-
type [, int precision [, int scale [, int size[]]]]]) )
```

Table 10.5 describes the parameters of function **ibm\_db.bind\_param**

Name	Description
<b>stmt</b>	A prepared statement returned from <b>ibm_db.prepare()</b>
<b>Parameter-number</b>	Specifies the 1-indexed position of the parameter in the prepared statement.
<b>variable</b>	A Python variable to be bound to the parameter specified by parameter-number
<b>Parameter-type</b>	Optional: The type of the parameter marker. It can be an input parameter (SQL_PARAM_INPUT), an output parameter (SQL_PARAM_OUTPUT), or as a parameter that accepts input and returns output (SQL_PARAM_INPUT_OUTPUT). To avoid memory overhead, you can also specify PARAM_FILE to bind the Python variable to the name of a file that contains large object (BLOB, CLOB, or DBCLOB) data. By default, it's SQL_PARAM_INPUT.
<b>data-type</b>	Optional: A constant specifying the SQL data type that the Python variable should be bound as: one of SQL_BINARY, DB2_CHAR, DB2_DOUBLE, or DB2_LONG
<b>precision</b>	Optional: Specifies the precision that the variable should be bound to the database.

<b>scale</b>	Optional: Specifies the scale that the variable should be bound to the database
<b>size</b>	Optional: Specifies the size that should be retrieved from an INOUT/OUT parameter

**Table 10.5 - Parameters for function `ibm_db.bind_param`**

The function returns `True` if the bindings are successful, otherwise it raise an exception if failure.

For example, you can call `ibm_db.bind_param` to bind the value "A00" to the parameter marker in the `SELECT` statement prepared in step 2 as follows:

```
ibm_db.bind_param(stmt, 1, "A00")
```

After successful execution, the value "A00" would be bound to the parameter marker (?) in the SQL's predicate "workdept = ?".

4. Execute the SQL statement by calling `ibm_db.execute()` function.

The syntax of function `ibm_db.execute` is as follows:

```
Py_True/Py_False ibm_db.execute(IBM_DBStatement stmt [, tuple
parameters])
```

Table 10.6 describes the parameters of `ibm_db.execute()`

Name	Description
<b>stmt</b>	A prepared statement returned from <code>ibm_db.prepare()</code>
<b>parameters</b>	Optional: A tuple of input parameters matching any parameter markers contained in the prepared statement. It's optional if you have called the function <code>bind_param</code> to bind the parameters.

**Table 10.6 - Parameters for function `ibm_db.execute`**

The function returns `True` if the SQL is executed successfully, otherwise it raises an exception. If the SQL to run is a `SELECT` statement or a `CALL` to a stored procedure (calling stored procedures will be discussed more in later sections) that returns one or more result sets, the result sets will be associated with the statement object. You can then use various fetch functions, such as `ibm_db.fetch_tuple`, to retrieve rows from the result set as discussed in the previous *section 10.3.4*.

For example, you can execute the `SELECT` statement prepared in previous steps as follows:

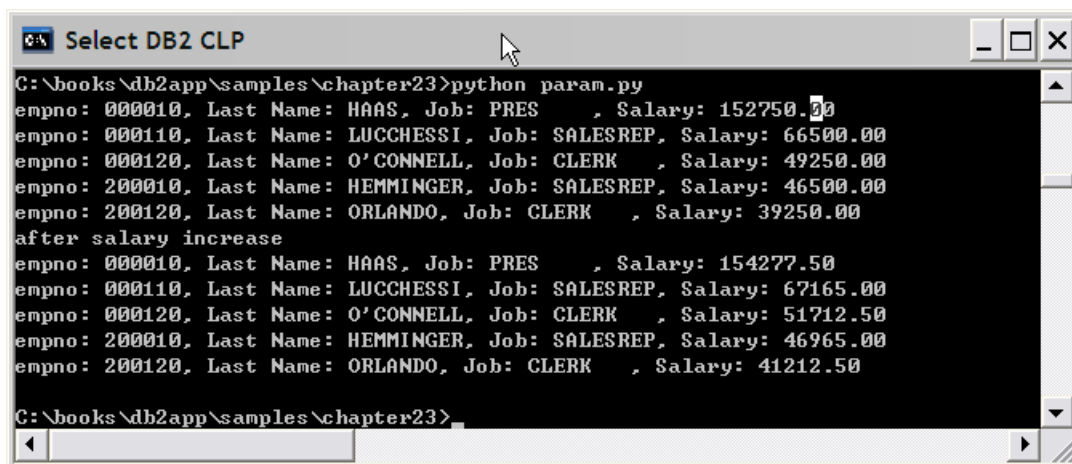
```
ibm_db.execute(stmt)
```

After successful execution of the SQL statement, a result set is associated with the statement. You can then begin fetching rows from the result set.

The above code snippets are part of the script `param.py` included in the exercise files accompanying this book. In addition to **SELECT** statements, the script also contains an example of executing an **UPDATE** statement with parameter markers. If you want to test the code snippets, you can modify the user ID and password in the script appropriately and run it as:

```
python param.py
```

The result of executing the script `param.py` is illustrated in *Figure 10.7*.



```
C:\books\db2app\samples\chapter23>python param.py
empno: 000010, Last Name: HAAS, Job: PRES      , Salary: 152750.00
empno: 000110, Last Name: LUCCHESSI, Job: SALESREP, Salary: 66500.00
empno: 000120, Last Name: O'CONNELL, Job: CLERK  , Salary: 49250.00
empno: 200010, Last Name: HEMMINGER, Job: SALESREP, Salary: 46500.00
empno: 200120, Last Name: ORLANDO, Job: CLERK  , Salary: 39250.00
after salary increase
empno: 000010, Last Name: HAAS, Job: PRES      , Salary: 154277.50
empno: 000110, Last Name: LUCCHESSI, Job: SALESREP, Salary: 67165.00
empno: 000120, Last Name: O'CONNELL, Job: CLERK  , Salary: 51712.50
empno: 200010, Last Name: HEMMINGER, Job: SALESREP, Salary: 46965.00
empno: 200120, Last Name: ORLANDO, Job: CLERK  , Salary: 41212.50
C:\books\db2app\samples\chapter23>
```

Figure 10.7 - Executing the script `param.py`

### 10.3.5 Call a stored procedure

To call a stored procedure from a Python application, you can follow the steps below:

1. Create a database connection object by connecting to the database with the `ibm_db.connect` function.
2. Invoke the stored procedure by calling the function `ibm_db.callproc`, which returns a tuple with the first item as the statement object and the rest of items as the modified copy of the parameters. The statement object contains the result sets returned from a stored procedure if there is any, while the `OUT` and `INOUT` parameters are replaced with possibly new values, and the input parameters are left unchanged. If there is multiple result sets, you can call `ibm_db.next_result` method by passing the original statement resource as the first argument, for example `stmt1 = ibm_db.next_result(stmt)`, to move to the next result set.

The syntax of `ibm_db.callproc` is:

```
tuple ibm_db.callproc ( IBM_DBConnection connection, string procname,
[,tuple parameters] )
```

*Table 10.7* describes the parameters of `ibm_db.callproc`

Name	Description
<code>connection</code>	A valid database connection object as returned from <code>ibm_db.connect()</code>
<code>procname</code>	The name of the stored procedure to call
<code>parameters</code>	Optional: the tuple of parameters must contain one entry for each parameter that the procedure expects. It's optional if the stored procedure has no parameter.

**Table 10.7 - Parameters for function `ibm_db.callproc`**

**Note:**

At the time of writing, the function `ibm_db.callproc` is a new function to add support for calling a stored procedure. You should use this function to call any store procedure. The functions `prepare/bind_param/execute` can also be used to call a stored procedure with `IN` parameters via the `CALL` statement.

For example, say you have a SQL procedure `sp_get_employees` as defined in *Listing 10.8*.

```
CREATE PROCEDURE sp_get_employees (IN dept_no CHAR(3), OUT dept_name
VARCHAR(36))
DYNAMIC RESULT SETS 1
BEGIN
  DECLARE emp_cursor CURSOR WITH RETURN TO CLIENT FOR SELECT firstnme,
    lastname FROM employee WHERE workdept=dept_no;
  OPEN emp_cursor;
  SELECT deptname INTO dept_name FROM department WHERE deptno=dept_no;
END @
```

**Listing 10.8 – `sp_get_employees.db2`**

The above SQL procedure has two parameters, `dept_no` is the input parameter and `dept_name` is the output parameter, and it is also to return one result set as defined in the cursor `emp_cursor` to the client.

You can then call the stored procedure as illustrated in *Listing 10.9*.

```
deptno='A00'
deptname=''
```

```
(1) stmt, deptno,deptname=ibm_db.callproc( conn, "sp_get_employees",
    (deptno, deptname))
(2) mytuple = ibm_db.fetch_tuple(stmt)
    while mytuple != False:
        print "First Name is : %s, Last Name is : %s" % (mytuple[0],
            mytuple[1])
        mytuple = ibm_db.fetch_tuple(stmt)
```

**Listing 10.9 – calling function `ibm_db.callproc`**

Where:

- (1) `deptno` is the Python variable containing the input value for the parameter `dept_no` of the stored procedure, and `deptname` is the variable to contain the output value of the parameter `dept_name`. After successful execution, the function returns a statement object in the variable `stmt`, and the variable `deptname` is modified to have the value of the output parameter while the input variable `deptno` is untouched. Since the SQL procedure is also returning one result set, the statement object will contain the result set returned from the SQL procedure.
- (2) Fetch the rows from the result set associated with the statement object as you have done in previous sections.

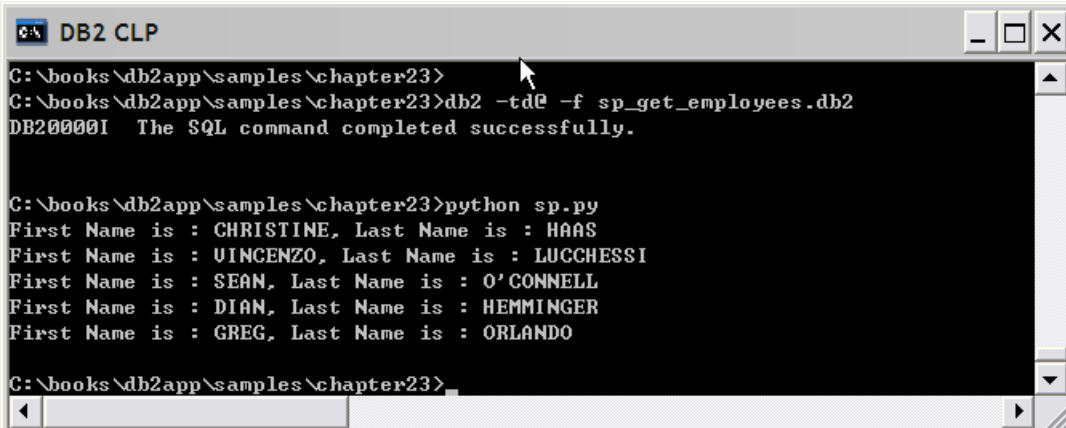
Above code snippets are part of the script `sp.py` accompanying the book. If you want to test the code snippets, you need to run the following commands to first create the SQL procedure `sp_get_lastname` as defined in the db2 script `sp_get_employees.db2` accompanying the book:

```
db2 connect to sample
db2 -td@ -f sp_get_employees.db2
db2 terminate
```

You should also modify the user ID and password in the `sp.py` script appropriately and run it as:

```
python sp.py
```

The result of executing the script `sp.py` is illustrated in *Figure 10.8*:



```
DB2 CLP
C:\books\db2app\samples\chapter23>
C:\books\db2app\samples\chapter23>db2 -td@ -f sp_get_employees.db2
DB20000I The SQL command completed successfully.

C:\books\db2app\samples\chapter23>python sp.py
First Name is : CHRISTINE, Last Name is : HAAS
First Name is : UINCENZO, Last Name is : LUCCHESSI
First Name is : SEAN, Last Name is : O'CONNELL
First Name is : DIAN, Last Name is : HEMMINGER
First Name is : GREG, Last Name is : ORLANDO

C:\books\db2app\samples\chapter23>
```

Figure 10.8 - Executing the script `sp.py`

## 10.4 Exercises

In this exercise, you will practice writing a small Python script to access data in the `SAMPLE` database.

1. Log on to the server as the instance owner (for example `db2inst1` on Linux or `db2admin` on Windows)
2. Run the following command to create the `SAMPLE` database if you haven't done so before: `db2samp1`
3. Write a Python script to print out all the employees who are working in the department of "SOFTWARE SUPPORT". At the same time increase the salary by 5% for each employee who was hired before '1996-01-01' in this department.
4. If you have problems creating this script, the solution is provided in the script `python_ex1.py` accompanying the book. You can modify the user ID and password in the script appropriately and test it out as follows:

```
python python_ex1.py
```

## 10.5 Summary

In this chapter, we have discussed different types of Python DB2 drivers/adapters and APIs that you can use to access DB2 data servers from Python applications. Details are also provided on how to execute SQL statements and call stored procedures from Python applications by calling the `ibm_db` driver/APIs.

## 10.6 Review questions

1. What are the drivers or adapters that a Python application can use to access a DB2 database?

2. What's the main difference between `ibm_db` and `ibm_db_dbi` drivers?
3. Which drivers or adapters support using Python functions and expressions to construct a query, instead of using normal SQL statements?
4. Which of the following APIs provides the best support for DB2 advanced features, such as using pureXML and accessing metadata?
  - A. `ibm_db`
  - B. `ibm_db_dbi`
  - C. `ibm_db_sa`
  - D. `ibm_db_django`
  - E. None of the above
5. Which of the following are the steps you can use to execute a SQL with parameter marker?
  - A. `prepare`, `bind_param`, `execute`
  - B. `prepare`, `bind_variable`, `exec_immediate`
  - C. `prepare`, `bind_param`, `exec_immediate`
  - D. `prepare`, `execute`, `fetchrow`
  - E. None of the above
6. Which of the following functions can be used to execute a SQL directly without first preparing it?
  - A. `execute`
  - B. `exec_immediate`
  - C. `execute_immediate`
  - D. `execute_imm`
  - E. None of the above
7. Which of the following functions can be used to call a stored procedure?
  - A. `exec_proc`
  - B. `execute_call`
  - C. `callproc`
  - D. `procall`
  - E. None of the above







## Appendix A – Solutions to the review questions

### Chapter 1

1. Stored procedures improve performance because they reduce network traffic.
2. Users can extend the SQL language by developing User-defined functions (UDFs)
3. CLI is a superset of ODBC
4. Static SQL knows the entire SQL statement at precompile time. Dynamic SQL will get this information at runtime.
5. Type 2 requires a DB2 client to be installed
6. D. All of the above
7. E. All of the above
8. C. ODBC and JDBC always use dynamic SQL
9. C. DB2 .NET Data provider
10. D. ibm\_db\_python

### Chapter 2

1. Relational database technology has been available for close to 30 years. The technology is very robust, reliable, secure and good for performance to retrieve information. XML is data like other data, so it makes sense to store it in a database, but making some changes/modifications to the database engine so it can handle XQuery.
2. The two types are: XML-enabled databases ("old" technology), and Native XML databases (like DB2)
3. pureXML characteristics: (1) XML has been stored in the database in hierarchical format (which is the format of XML documents). (2) There is a second part of the DB2 engine that can handle XML natively using XQuery/XPath.

4. Normally your code will be smaller which means there would be less instructions to execute, and less instructions to maintain. Your code is smaller because there is no need to do any parsing in your code to build a tree in order to navigate through an XML document. The parsing has been done when the XML document was first inserted into the database.
5. There are two ways: (1) Simply use a SQL INSERT statement, where you pass the XML document in single quotes. (2) Use the DB2 IMPORT utility when you want to insert from a file.
6. E. All of the above. SQL by itself, can also retrieve XML data, only that it would retrieve the entire XML document. If you only want part of it, then you must use SQL/XML or XQuery.
7. E. XMLNAVIGATE
8. C. XMLQUERY is not an XQuery function but an SQL/XML function
9. C. Use the TRANSFORM expression
10. E. None of the above. XML indexes can also be created for values. An XML Schema repository is stored inside the DB2 database, and an XML document can be validated with a BEFORE trigger.

### Chapter 3

1. The benefits of stored procedures are:
  - Reduces network traffic; therefore allow for better performance
  - Centralizes logic in the database, therefore generic stored procedures can be written that can be used by many client applications
  - Allows users to perform operations on objects they don't have explicit access to. Through the stored procedure, there is controlled access.
2. No. Scalar UDFs cannot perform UPDATE operations. For such operations, use a TABLE function.
3. You can invoke a scalar UDF in two ways:
  - Using values statement
  - Using a SELECT statement
4. No. A BEFORE trigger cannot UPDATE. Before triggers can be used for checking validity of your data before INSERTs. For UPDATE operations as part of the trigger action, use AFTER triggers.

5. The SPECIFIC keyword provides a unique name to a stored procedure. Stored procedures can be overloaded, and using this unique name can help manage the procedures.
6. B. Present. This is not a valid trigger
7. D. IBM Data Studio cannot be used to create triggers.
8. E. Both C and D are valid. The 'AS' keyword is optional.
9. B. XML is used as the underlying technology for Web services regardless of whether they are SOAP or REST based.
10. B. Starting with DB2 9.7, UDFs and Triggers have full SQL PL support. Prior to DB2 9.7, they only supported inline SQL PL, a subset of the SQL PL language.

#### Chapter 4

1. JDBC is the standard to access databases using Java for dynamic SQL. SQLJ is the standard for embedded static SQL statements.
2. JDBC Type 2 and Type 4
3. db2jcc.jar is the driver file that is JDBC 3.0 specification compliant, while db2jcc4.jar supports part of JDBC 4.0 specification and earlier.
4. An iterator in SQLJ is equivalent to a result set in JDBC. It returns several rows that can be processed on a loop by the program.
5. A default context is a context that would be used by default when not specified in a SQLJ program. For example, if you specify that 'ctx1' is the default connection context, and later on you don't provide it in your statements, then this default context is used.
6. E. None of the above. db2jcc.jar and db2jcc4.jar both include a JDBC Type 2 and JDBC Type 4 driver
7. D. All of the above. pureQuery can also be used even if an ORM is in place. It compliments the ORM.
8. D. JDBC, SQLJ, and pureQuery can be combined
9. D. ResultSetStatement is not a valid method.
10. A. Executive context does not exist. It should be execution context.

#### Chapter 5

1. CLI/ODBC is an alternative to embedded dynamic SQL, but unlike embedded SQL, it does not require host variables or a precompiler. Applications can be run

against a variety of databases without having to be compiled against each of these databases.

2. A cursor is a named control structure used by an application program to point to a specific row within an ordered set of rows.
3. A parameter marker, denoted by a question mark (?), is a place holder in an SQL statement whose value is obtained during statement execution.
4. An ODBC/CLI handle is a pointer to a variable which is used for passing references to the variable between parts of the program.
5. SQL\_SUCCESS  
SQL\_SUCCESS\_WITH\_INFO  
SQL\_INVALID\_HANDLE  
SQL\_ERROR
6. C. File extension of embedded C++ application on windows will be .sqx
7. B. db2bfd -s is the command that can dump the SQL statements from a bind file
8. C. db2 list system odbc data sources
9. A. SQLAllocHandle () API is used for allocating all the handles.
10. A. Allocate environment handle -> allocate connection handle -> allocate statement handle -> free statement handle-> free connection handle -> free environment handle is the correct flow of handle allocations.

## Chapter 6

1. 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
2. In Step 1 for the connectivity settings, you need to additionally catalog the DB2 database as an ODBC data source.
3. Data Access in ADO.NET relies on two components: DataSet and Data Provider.

### DataSet

The dataset is a disconnected, in-memory representation of data. It can be considered as a local copy of the relevant portions of the database. The DataSet is persisted in memory and the data in it can be manipulated and updated independent of the database. When the use of this DataSet is finished, changes can be made back to the central database for updating. The data in DataSet can be loaded from any valid data source like DB2.

### Data Provider

---

The Data Provider is responsible for providing and maintaining the connection to the database. A DataProvider is a set of related components that work together to provide data in an efficient and performance driven manner. Each DataProvider consists of component classes.

4. You can run 32-bit .NET applications on a 64-bit Windows instance, using a 32-bit edition of the IBM Data Server Provider for .NET. To get a 32-bit IBM Data Server Provider for .NET on your 64-bit computer, you can install the 32-bit version of IBM Data Server Driver Package.
5. To use the IBM Database Add-Ins for Visual Studio, download it from <http://www.ibm.com/db2/express/download.html>. After you install a DB2 product, install the IBM Database Add-Ins for Visual Studio by double clicking on the executable `db2exc_vsai_XXX_WIN_x86.exe`, where `XXX` represents a version number that matches the version number of the DB2 server.
6. A
7. C
8. D
9. B
10. C

## Chapter 7

1. Ruby on Rails is a popular framework for developing Web applications. It is based on the MVC architecture, and follows several philosophies such as "Convention over configuration" and "Don't repeat yourself" which allows developers to quickly deliver nice applications.
2. `ibm_db` is a gem that includes the Ruby driver and Rails adapter for DB2. It can be installed using this command:  

```
gem install ibm_db
```
3. Rails' support for DB2 is provided through the `ibm_db` gem. This gem contains a driver (written in C) that allows Ruby to communicate with DB2, and an adapter written in Ruby that enables ActiveRecord to work with DB2. ActiveRecord is the ORM layer that maps object oriented classes to relational tables.
4. This file allows you to configure your connection to a DB2 database by specifying the host name, port, user ID, password, adapter name, and so on.
5. Yes, we can. DB2 enables the use of XQuery, so any data in your XML document is easily accessible.
6. C. The older versions of the same library must be deleted before the new one is installed
7. A. Ruby. A gem is a standardized package format. Ruby is the programming language.

8. C. IBM\_DB adapter utilizes the IBM Driver for **ODBC** and **CLI** to connect to IBM data servers.
9. A. The **Account** is optional.
10. B.

## Chapter 8

1. As the name suggests, the non persistent connection disconnects and frees up the connection resources after each `db2_close`, or connection resource is set to NULL, or the script ends.

In the case of persistent connections, the connection resources are not freed up after a `db2_close` or the script is exited. Whenever a new connection is requested, PHP tries to reuse the connection with the same credentials

2. Connection pooling helps database application performance due to the reduction of new connections by reusing the old connections established before. It can be beneficial when the connections made have a short duration.
3. Enable PHP support in Apache HTTP Server 2.x by adding the following lines to the `httpd.conf` file, where `phpdir` refers to the PHP install directory:

```
LoadModule php5_module 'phpdir/php5apache2.dll'  
AddType application/x-httpd-php .php  
PHPIniDir 'phpdir'
```

4. To determine the configuration file path issue the `php -i` command and look for the `php.ini` keyword
5. The **ibm\_db2** extension API makes porting an application that was previously written for Unified ODBC almost as easy as globally changing the "`odbc_`" function name to "`db2_`" throughout the source code of your application.
6. D
7. B
8. D
9. C
10. A

## Chapter 9

1. Perl Database Interface (DBI)

2. The standard Perl DBI provides the database interface to the Perl applications, while in the background it will load the DBD drivers and the DBD drivers are the ones who actually do the work on behalf of databases.
3. C
4. A
5. C

### **Chapter 10**

1. `ibm_db`, `ibm_db_dbi`, `ibm_db_sa`, `ibm_db_django`
2. `ibm_db` implements a set of proprietary database APIs defined by IBM itself, while `ibm_db_dbi` is a wrapper built upon `ibm_db` driver to support the standard Python Database API specification
3. `ibm_db_sa` and `ibm_db_django`
4. A
5. A
6. B
7. C

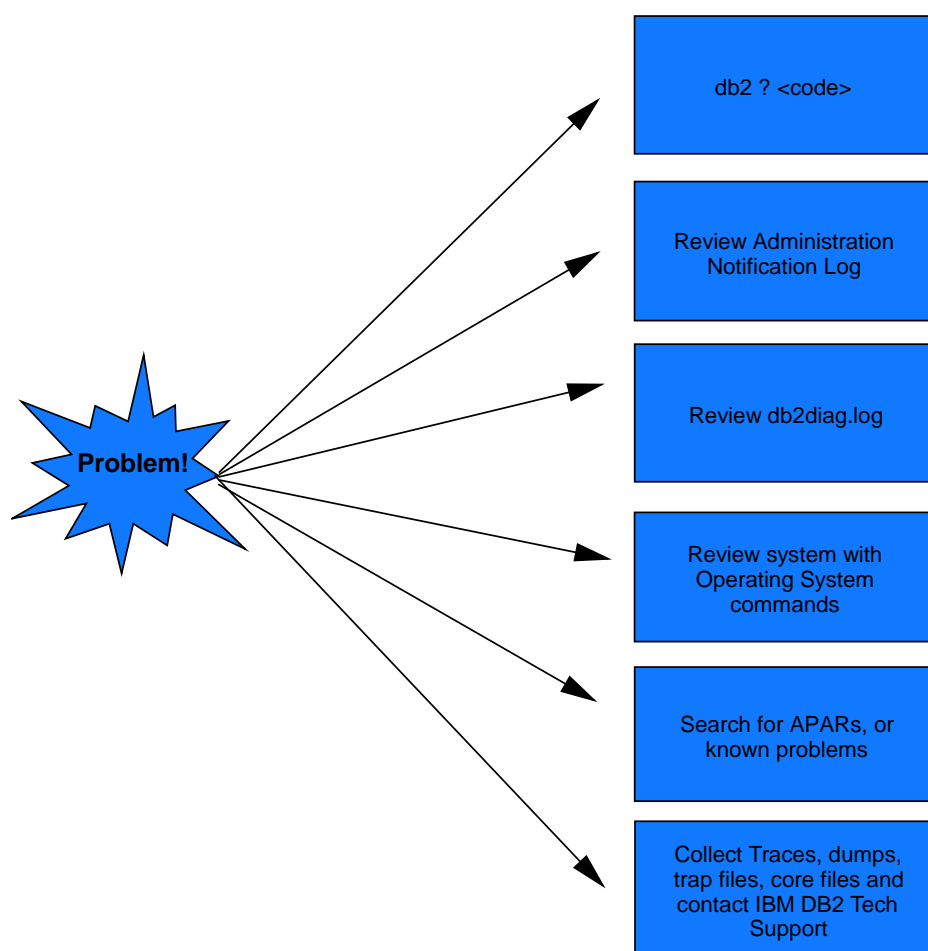




# B

## Appendix B – Troubleshooting

This appendix discusses how to troubleshoot problems that may be encountered when working with DB2. *Figure B.1* provides a brief overview of the actions to take should a problem arise.



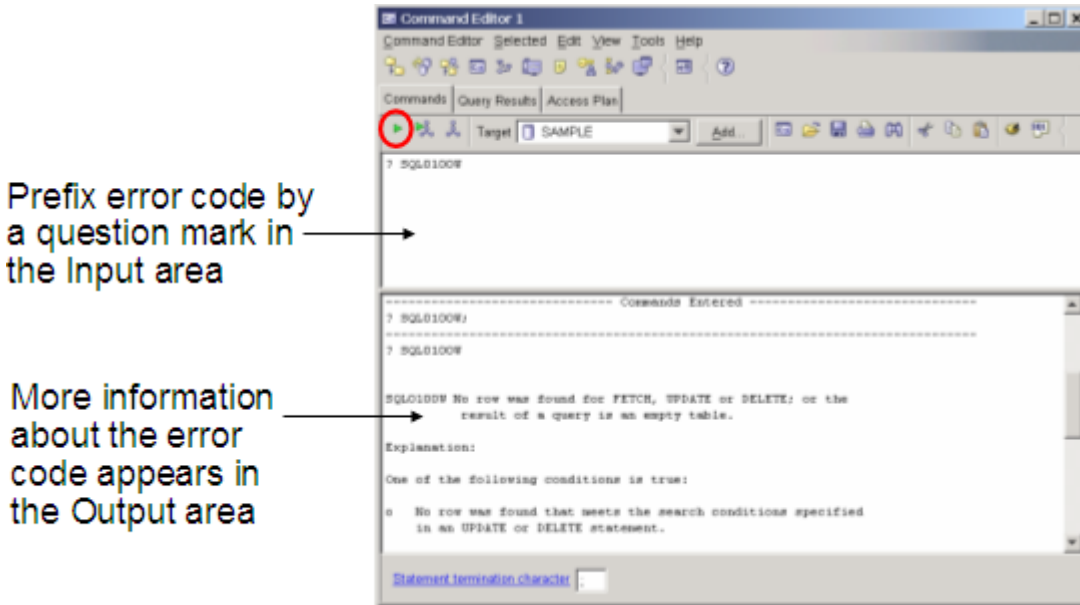
**Figure B.1 – Troubleshooting overview**

**Note:**

For more information about troubleshooting, watch this video:  
<http://www.channeldb2.com/video/video/show?id=807741:Video:4462>

## B.1 Finding more information about error codes

To obtain more information about an error code received, enter the code prefixed by a question mark in the Command Editor input area and click the *Execute* button. This is shown in *Figure B.2*.



**Figure B.2 – Finding more information about DB2 error codes**

The question mark (?) invokes the DB2 help command. Below are several examples of how to invoke it for help if you receive, for example, the SQL error code “-104”. All of the examples below are equivalent.

```
db2 ? SQL0104N
db2 ? SQL104N
db2 ? SQL-0104
db2 ? SQL-104
db2 ? SQL-104N
```

## B.2 SQLCODE and SQLSTATE

An SQLCODE is a code received after every SQL statement is executed. The meanings of the values are summarized below:

- SQLCODE = 0; the command was successful
- SQLCODE > 0; the command was successful, but returned a warning
- SQLCODE < 0; the command was unsuccessful and returned an error

The SQLSTATE is a five-character string that conforms to the ISO/ANSI SQL92 standard. The first two characters are known as the SQLSTATE class code:

- A class code of 00 means the command was successful.
- A class code of 01 implies a warning.
- A class code of 02 implies a not found condition.
- All other class codes are considered errors.

### B.3 DB2 Administration Notification Log

The DB2 administration notification log provides diagnostic information about errors at the point of failure. On Linux and UNIX platforms, the administration notification log is a text file called <instance name>.nfy (e.g. "db2inst.nfy"). On Windows, all administration notification messages are written to the Windows Event Log.

The DBM configuration parameter `notifylevel` allows administrators to specify the level of information to be recorded:

- 0 -- No administration notification messages captured (not recommended)
- 1 -- Fatal or unrecoverable errors
- 2 -- Immediate action required
- 3 -- Important information, no immediate action required (the default)
- 4 -- Informational messages

### B.4 db2diag.log

The db2diag.log provides more detailed information than the DB2 Administration notification log. It is normally used only by IBM DB2 technical support or experienced DBAs. Information in the db2diag.log includes:

- The DB2 code location reporting an error.
- Application identifiers that allow you to match up entries pertaining to an application on the db2diag.logs of servers and clients.
- A diagnostic message (beginning with "DIA") explaining the reason for the error.
- Any available supporting data, such as SQLCA data structures and pointers to the location of any extra dump or trap files.

On Windows (other than Vista), the db2diag.log is located by default under the directory:

```
C:\Documents and Settings\All Users\Application  
Data\IBM\DB2\DB2COPY1\<>instance name<
```

On Windows Vista, the db2diag.log is located by default under the directory:

```
C:\ProgramData\IBM\DB2\DB2COPY1\<>instance name<
```

On Linux/UNIX, the db2diag.log is located by default under the directory:

```
/home/<instance_owner>/sqlllib/db2dump
```

The verbosity of diagnostic text is determined by the dbm cfg configuration parameter DIAGLEVEL. The range is 0 to 4, where 0 is the least verbose, and 4 is the most. The default level is 3.

## B.5 CLI traces

For CLI, Java, PHP, and Ruby on Rails applications, you may turn on the CLI trace facility to troubleshoot your application. This can be done by making changes to the db2cli.ini file at the server where your application is running. Typical entries in the db2cli.ini file are shown below in *Listing B.1*.

```
[common]
trace=0
tracerefreshinterval=300
tracepathname=/path/to/writeable/directory
traceflush=1
```

### Listing B.1 - db2cli.ini file entries to turn on CLI Tracing

Low level tracing (db2trc) is also available, but this is typically only useful for DB2 technical support.

## B.6 DB2 Defects and Fixes

Sometimes a problem you encounter may be caused by a defect in DB2. IBM regularly releases fix packs which contain code fixes for defects (APARs). The fix pack documentation contains a list of the fixes contained in the fix pack. When developing new applications, we always recommend using the latest fix pack to benefit from the latest fixes. To view your current version and fix pack level: from the Control Center, select the **About** option from the **Help** menu; or from the Command Window, type `db2level`. Note that fix packs and official IBM DB2 technical support are not offered with DB2 Express-C, With DB2 Express-C, fixes are incorporated into the image itself rather than applied as fix packs.

## References

- [1] ZIKOPOULOS, P. *IBM® DB2® Universal Database™ and the Microsoft® Excel Application Developer... for Beginners*, dbazine.com article, April 2005 <http://www.dbazine.com/db2/db2-disarticles/zikopoulos15>
- [2] ZIKOPOULOS, P. *DB2 9 and Microsoft Access 2007 Part 1: Getting the Data...*, Database Journal article, May 2008 <http://www.databasejournal.com/features/db2/article.php/3741221>
- [3] BHOGAL, K. *Use Microsoft Access to interact with your DB2 data*, developerWorks article, May 2006. <http://www.ibm.com/developerworks/db2/library/techarticle/dm-0605bhogal/>
- [4] CHUN, J., CIRONE P. *DB2 packages: Concepts, examples, and common problems*, developerWorks article, June 2006 <http://www.ibm.com/developerworks/data/library/techarticle/dm-0606chun/index.html>
- [5] CHEN Whei-Jen et al. *DB2 Express-C: The Developer Handbook for XML, PHP, C/C++, Java, and .NET* August 2006 - SG24-7301-00 <http://www.redbooks.ibm.com/abstracts/sq247301.html?Open>

## Resources

### Web sites

1. DB2 Express-C web site:  
[www.ibm.com/db2/express](http://www.ibm.com/db2/express)  
Use this web site to download the image for DB2 Express-C servers, DB2 clients, DB2 drivers, manuals, access to the team blog, mailing list sign up, etc.
2. DB2 Express-C forum: [www.ibm.com/developerworks/forums/dw\\_forum.jsp?forum=805&cat=19](http://www.ibm.com/developerworks/forums/dw_forum.jsp?forum=805&cat=19)  
Use the forum to post technical questions when you cannot find the answers in the manuals yourself.
3. DB2 Information Center  
<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp>  
The information center provides access to the online manuals. It is the most up to date source of information.
4. developerWorks  
<http://www-128.ibm.com/developerworks/db2>  
This Web site is an excellent resource for developers and DBAs providing access to current articles, tutorials, etc. for free.
5. alphaWorks®  
<http://www.alphaworks.ibm.com/>  
This Web site provides direct access to IBM's emerging technology. It is a place where one can find the latest technologies from IBM Research.

6. planetDB2

[www.planetDB2.com](http://www.planetDB2.com)

This is a blog aggregator from many contributors who blog about DB2.

7. DB2 Technical Support

If you purchased the 12 months subscription license of DB2 Express-C, you can download fixpacks from this Web site.

[http://www.ibm.com/software/data/db2/support/db2\\_9/](http://www.ibm.com/software/data/db2/support/db2_9/)

8. ChannelDB2

ChannelDB2 is a social network for the DB2 community. It features content such as DB2 related videos, demos, podcasts, blogs, discussions, resources, etc. for Linux, UNIX, Windows, z/OS, and i5/OS.

<http://www.ChannelDB2.com/>

## Books

1. Free Redbook: DB2 Express-C: The Developer Handbook for XML, PHP, C/C++, Java, and .NET  
Whei-Jen Chen, John Chun, Naomi Ngan, Rakesh Ranjan, Manoj K. Sardana,  
August 2006 - SG24-7301-00  
<http://www.redbooks.ibm.com/abstracts/sg247301.html?Open>
2. Understanding DB2 – Learning Visually with Examples V9.5  
Raul F. Chong, et all. January 2008  
ISBN-10: 0131580183
3. DB2 9: pureXML overview and fast start by Cynthia M. Saracco, Don Chamberlin, Rav Ahuja  
June 2006 SG24-7298  
<http://www.redbooks.ibm.com/abstracts/sg247298.html?Open>
4. DB2® SQL PL: Essential Guide for DB2® UDB on Linux™, UNIX®, Windows™, i5/OS™, and z/OS®, 2nd Edition  
Zamil Janmohamed, Clara Liu, Drew Bradstock, Raul Chong, Michael Gao, Fraser McArthur, Paul Yip  
ISBN: 0-13-100772-6
5. Free Redbook: DB2 pureXML Guide  
Whei-Jen Chen, Art Sammartino, Dobromir Goutev, Felicity Hendricks, Ipepei Komi, Ming-Pang Wei, Rav Ahuja, Matthias Nicola. August 2007  
<http://www.redbooks.ibm.com/abstracts/sg247315.html?Open>
6. Information on Demand - Introduction to DB2 9 New Features  
Paul Zikopoulos, George Baklarz, Chris Eaton, Leon Katsnelson  
ISBN-10: 0071487832  
ISBN-13: 978-0071487832

## Contact emails

General DB2 Express-C mailbox: [db2x@ca.ibm.com](mailto:db2x@ca.ibm.com)

General DB2 on Campus program mailbox: [db2univ@ca.ibm.com](mailto:db2univ@ca.ibm.com)

**Getting started with DB2 application development couldn't be easier.**

**Read this book to:**

- Discover DB2<sup>®</sup> application development using DB2 Express-C
- Write SQL, XQuery, and understand pureXML<sup>®</sup> technology
- Learn how to develop DB2 stored procedures, functions and data Web services
- Learn how to work with DB2 and Java<sup>™</sup>, C/C++, .NET, PHP, Ruby on Rails, Perl, and Python
- Troubleshoot DB2 database-related problems
- Practice with hands-on exercises

DB2 Express-C from IBM is the no-charge edition of DB2 data server for managing relational and XML data with ease. No-charge means DB2 Express-C is free to download, free to develop your applications, free to deploy into production, and even free to embed and distribute with your solution. And, DB2 does not place any artificial limits on the size of databases, number of databases, or number of users.

DB2 Express-C runs on Windows<sup>®</sup>, Linux<sup>®</sup>, Solaris, and Mac OS X systems, and provides application drivers for a variety of programming languages and frameworks including C/C++, Java, .NET, Ruby on Rails, PHP, Perl, and Python. If you require even greater scalability or more advanced functionality, you can seamlessly deploy applications built using DB2 Express-C to other DB2 editions such as DB2 Workgroup and DB2 Enterprise.

This free edition of DB2 is ideal for developers, consultants, ISVs, DBAs, students, or anyone who intends to develop, test, deploy, or distribute database applications. Join the growing DB2 Express-C user community today and take DB2 Express-C for a test drive. Start discovering how you can create next generation applications and deliver innovative solutions.

To learn more or download DB2 Express-C, visit [ibm.com/db2/express](http://ibm.com/db2/express)

To socialize and watch related videos, visit [channelDB2.com](http://channelDB2.com)

This book is part of the DB2 on Campus book series, free eBooks for the community. Learn more at [db2university.com](http://db2university.com)



**Price: 24.99USD**