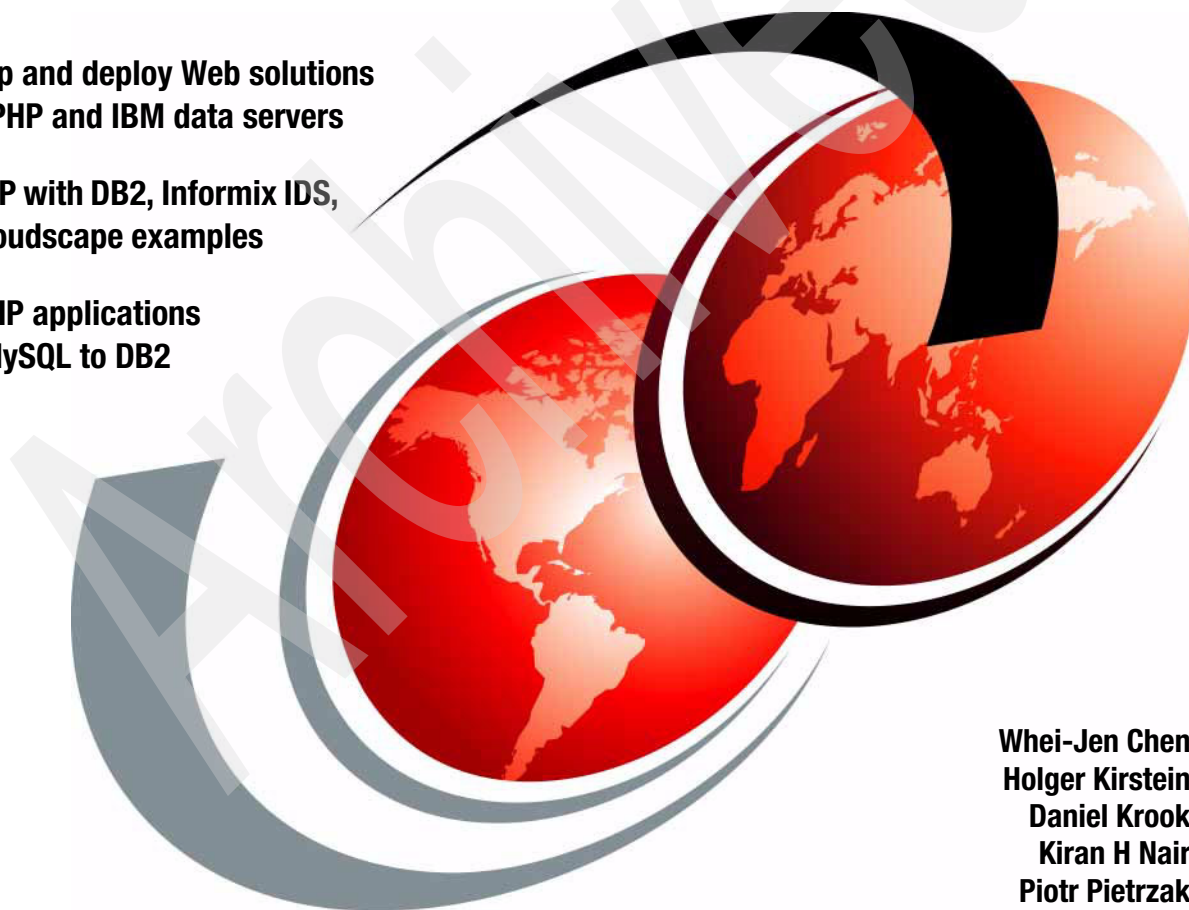


# Developing PHP Applications for IBM Data Servers

Develop and deploy Web solutions using PHP and IBM data servers

See PHP with DB2, Informix IDS, and Cloudscape examples

Port PHP applications from MySQL to DB2



Whei-Jen Chen  
Holger Kirstein  
Daniel Krook  
Kiran H Nair  
Piotr Pietrzak





International Technical Support Organization

## **Developing PHP Applications for IBM Data Servers**

May 2006

Archived

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

### **First Edition (May 2006)**

This edition applies to DB2 UDB Version 8.2, Informix IDS Version 10, PHP Versions 4 and 5, Apache 1.3, and Apache 2.

**© Copyright International Business Machines Corporation 2006. All rights reserved.**

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Figures</b> .....	vii
<b>Tables</b> .....	ix
<b>Notices</b> .....	xi
Trademarks .....	xii
<b>Preface</b> .....	xiii
The team that wrote this redbook .....	xiii
Acknowledgement .....	xv
Become a published author .....	xvi
Comments welcome .....	xvi
<b>Chapter 1. Technology overview</b> .....	1
1.1 Web application environment .....	2
1.1.1 Web application advantages .....	2
1.1.2 Web application challenges .....	3
1.1.3 The state of the Web application world .....	4
1.1.4 Web application components .....	4
1.2 IBM data servers .....	5
1.2.1 DB2 data server .....	5
1.2.2 Informix database server family .....	8
1.2.3 Cloudscape .....	10
1.3 HTTP Servers .....	11
1.3.1 Apache HTTP Server .....	12
1.3.2 IBM HTTP Server .....	15
1.3.3 Which Web server do I choose? .....	17
1.4 PHP .....	18
1.5 Database interfaces with PHP .....	22
1.5.1 DB2 and Cloudscape .....	22
1.5.2 Informix IDS database extensions .....	24
1.6 Zend products .....	27
<b>Chapter 2. Sample scenario description</b> .....	31
2.1 Application requirements .....	33
2.1.1 Customer interface functional requirements .....	35
2.1.2 Dealer interface functional requirements .....	45
2.1.3 Non-functional requirements of the entire application .....	52
2.2 Data model .....	53

2.2.1 Tables . . . . .	54
2.3 Application design . . . . .	56
2.3.1 Controller components . . . . .	57
2.3.2 View pages . . . . .	59
2.3.3 Domain objects . . . . .	60
2.3.4 Database adapters . . . . .	67
2.3.5 Conclusion . . . . .	71
2.4 Application installation . . . . .	72
<b>Chapter 3. Zend installation and configuration . . . . .</b>	<b>75</b>
3.1 Zend Core for IBM . . . . .	76
3.1.1 Installation: Linux . . . . .	76
3.1.2 Installation: Windows . . . . .	79
3.1.3 Sample application . . . . .	81
3.2 Zend Studio . . . . .	83
3.2.1 Installation . . . . .	83
3.2.2 Configuration . . . . .	85
3.2.3 Debugger . . . . .	89
<b>Chapter 4. PHP application development with DB2 . . . . .</b>	<b>95</b>
4.1 Application environment setup . . . . .	96
4.1.1 Lab environment description . . . . .	96
4.1.2 User IDs and group . . . . .	97
4.1.3 Database server installation and configuration . . . . .	98
4.1.4 Apache and PHP installation and configuration . . . . .	102
4.1.5 Environment verification . . . . .	110
4.2 Using PHP with DB2 database . . . . .	113
4.2.1 ibm_db2 . . . . .	113
4.2.2 PDO_IBM/PDO_ODBC . . . . .	139
4.2.3 Unified ODBC . . . . .	161
4.2.4 Porting PHP applications from Unified ODBC to ibm_db2 . . . . .	169
4.3 Troubleshooting DB2 - PHP applications . . . . .	170
4.3.1 Taking CLI trace . . . . .	170
4.3.2 db2diag.log . . . . .	174
4.3.3 Tools for monitoring, tuning, and troubleshooting . . . . .	175
4.3.4 Getting the best out of DB2 from PHP . . . . .	183
4.4 PHP application and DB2 for z/OS . . . . .	184
<b>Chapter 5. PHP applications with Informix database servers . . . . .</b>	<b>187</b>
5.1 Application environment setup . . . . .	188
5.1.1 Lab application environment . . . . .	189
5.1.2 Installing IBM Informix IDS V10.00 on Linux . . . . .	189
5.1.3 Installing the Apache Web server . . . . .	198
5.1.4 Installing the Informix client and connectivity . . . . .	202

5.1.5	Setting up PHP with multiple Informix IDS interfaces . . . . .	203
5.1.6	Installation verification . . . . .	217
5.2	Application development with Informix IDS . . . . .	227
5.2.1	Connection to the database . . . . .	227
5.2.2	Static and dynamic SQL statements in PHP . . . . .	232
5.2.3	Cursor and result set . . . . .	247
5.2.4	Complex data types . . . . .	269
5.2.5	Working with stored procedures and user defined functions . . . . .	276
5.2.6	BLOB and SBLOB data types . . . . .	303
5.2.7	Error handling . . . . .	315
5.2.8	Transactions and isolation level . . . . .	325
5.2.9	PHP and Informix XPS SQL extensions . . . . .	333
5.3	Optimization . . . . .	341
5.4	Apache, PHP, and Informix IDS on Windows . . . . .	347
5.4.1	Installation and configuration of Apache Web server . . . . .	347
5.4.2	Installation of the PHP and configuration with Apache . . . . .	348
5.4.3	Informix IDS, Informix connectivity, and PHP . . . . .	349
<b>Chapter 6.</b>	<b>Port PHP applications from MySQL V5 to DB2 UDB V8.2 . . . . .</b>	<b>353</b>
6.1	Introduction . . . . .	354
6.1.1	IBM migration offering . . . . .	354
6.2	Porting database server objects . . . . .	356
6.2.1	Stored procedures . . . . .	356
6.2.2	Porting MySQL stored procedures to DB2 . . . . .	359
6.2.3	Triggers . . . . .	364
6.2.4	Views . . . . .	367
6.2.5	User Defined Functions . . . . .	371
6.3	Porting a client PHP application . . . . .	371
6.3.1	Interface . . . . .	372
6.3.2	Prepared statements . . . . .	384
6.3.3	Transactions . . . . .	386
6.3.4	Stored procedures . . . . .	388
<b>Appendix A.</b>	<b>An introduction to Service Data Objects for PHP . . . . .</b>	<b>393</b>
A.1	Introduction . . . . .	394
A.2	SDO concepts . . . . .	395
A.3	Data Access Services . . . . .	397
A.3.1	Why choose Service Data Objects? . . . . .	398
A.4	Relationship to PHP Data Objects and SimpleXML . . . . .	399
A.5	Contact scenario . . . . .	400
A.5.1	Contact edit use case . . . . .	400
A.5.2	Retrieving the contact entry . . . . .	401
A.5.3	More on SDO navigation . . . . .	405

A.5.4 Modifying the data .....	407
A.5.5 More on SDO modification .....	409
A.6 Summary .....	411
<b>Appendix B. Additional material</b> .....	413
Locating the Web material .....	413
Using the Web material .....	413
System requirements for downloading the Web material .....	414
How to use the Web material .....	414
<b>Related publications</b> .....	415
IBM Redbooks .....	415
Other publications .....	415
Online resources .....	417
How to get IBM Redbooks .....	417
Help from IBM .....	417
<b>Index</b> .....	419



# Figures

1-1	Web application infrastructure . . . . .	5
1-2	Dynamic Web page generated by the PHP Hello World script . . . . .	20
2-1	Flow chart for Customer Portal of the Dealership application . . . . .	34
2-2	Flow diagram for Dealer Portal of the Dealership application . . . . .	35
2-3	The customer registration interface . . . . .	37
2-4	The Login interface . . . . .	38
2-5	Search vehicles page. . . . .	39
2-6	Promotions list interface. . . . .	40
2-7	The vehicle detail review interface. . . . .	41
2-8	Shows the Buy vehicle interface . . . . .	42
2-9	The Track order interface. . . . .	43
2-10	The Add review interface . . . . .	44
2-11	The dealer Login interface . . . . .	46
2-12	The dealer Inventory interface . . . . .	47
2-13	The Add vehicle model interface . . . . .	48
2-14	The dealer add Promotion interface. . . . .	50
2-15	The Orders interface . . . . .	51
2-16	The dealer Reports interface . . . . .	52
2-17	The Dealership database entities . . . . .	54
2-18	MVC architecture . . . . .	57
2-19	ControllerUtilities methods . . . . .	59
2-20	ViewUtilities methods. . . . .	60
2-21	ValueObject class tree . . . . .	61
2-22	Methods implemented by ValueObject and inherited by children . . . . .	62
2-23	An implementation only must declare its properties in its constructor . . . . .	63
2-24	Data access objects. . . . .	64
2-25	Abstract methods of DataAccessObject . . . . .	64
2-26	ReviewDAO extends DataAccessObject . . . . .	65
2-27	Database adapters defined . . . . .	67
2-28	DatabaseConnection properties . . . . .	67
2-29	DatabaseConnection methods. . . . .	68
2-30	A concrete implementation of a DatabaseConnection methods . . . . .	69
3-1	Sample installation window of Zend Core for IBM under Linux . . . . .	77
3-2	Zend Core for iBM Installation . . . . .	78
3-3	Default Zend Core Web site in IIS Manager window . . . . .	80
3-4	Zend Core for IBM GUI installed on Windows platform . . . . .	81
3-5	Sample application main window. . . . .	82
3-6	Zend Studio 5.1.0 installation window . . . . .	85

3-7	Set the preferences . . . . .	86
3-8	Project Properties . . . . .	87
3-9	SQL server icon . . . . .	88
3-10	Add connection icon. . . . .	88
3-11	Adding SQL server to the Zend Studio client. . . . .	89
3-12	Zend Studio Client Tools option window . . . . .	91
3-13	Running the debugger: Line being investigated. . . . .	92
3-14	Variables tab . . . . .	92
3-15	Watches debug window . . . . .	93
3-16	Debug Output window . . . . .	93
4-1	Selecting your wizard. . . . .	177
4-2	Create monitoring task. . . . .	178
4-3	Event monitors in Control Center. . . . .	179
4-4	Create Event Monitor. . . . .	180
4-5	Event Analyzer. . . . .	180
4-6	Access plan . . . . .	181
4-7	DB2 Design Advisor. . . . .	183
5-1	Compile options for PHP from the Red Hat distribution. . . . .	205
5-2	Configure options for the SLES9-based PHP library . . . . .	206
5-3	Informix PDO with phpinfo() . . . . .	218
5-4	Ifx interface settings with phpinfo() . . . . .	220
5-5	Configure command for unixODBC . . . . .	224
5-6	unixODBC PHP parameters . . . . .	225
5-7	General section for ODBC settings . . . . .	350
5-8	IBM Informix ODBC Driver Setup Connection . . . . .	351
A-1	The Role of a DAS . . . . .	394
A-2	SDO Model and Interfaces. . . . .	395
A-3	Example Person SDO instance and model . . . . .	396
A-4	The contact management main page . . . . .	401
A-5	Contact SDO instance . . . . .	404
A-6	The contact edit page . . . . .	407
A-7	The confirmation page . . . . .	409

# Tables

2-1	Customer use case: Register . . . . .	36
2-2	Customer use case: Login . . . . .	37
2-3	Customer use case: Search for vehicle . . . . .	38
2-4	Customer use case: See promotions . . . . .	40
2-5	Customer use case: See vehicle details . . . . .	41
2-6	Customer use case: Order vehicle . . . . .	42
2-7	Customer use case: Track order . . . . .	43
2-8	Customer use case: Add review . . . . .	43
2-9	Log out . . . . .	45
2-10	Login . . . . .	45
2-11	See inventory . . . . .	46
2-12	Add vehicle . . . . .	47
2-13	Add promotion . . . . .	49
2-14	See orders . . . . .	50
2-15	View reports . . . . .	51
4-1	Troubleshooting DB2, Apache, and PHP connection problems . . . . .	111
6-1	MySQL to DB2 interface migration options . . . . .	372
6-2	mysql to ibm_db2 function mapping . . . . .	374
6-3	MySQLi to ibm_db2 function mapping . . . . .	375
A-1	"contact" table definition . . . . .	400
A-2	"address" table definition . . . . .	401



# Notices

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

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

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

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

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

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

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

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

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

## COPYRIGHT LICENSE:

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

# Trademarks

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

@server®  
Redbooks (logo) ™  
developerWorks®  
eServer™  
iSeries™  
pSeries®  
xSeries®  
z/OS®  
zSeries®  
AIX®

AS/400®  
Cloudscape™  
Distributed Relational Database  
Architecture™  
DB2 Connect™  
DB2 Universal Database™  
DB2®  
DRDA®  
Informix®  
IBM®

OS/400®  
POWER™  
Rational®  
Redbooks™  
System i™  
System p™  
System z™  
WebSphere®

The following terms are trademarks of other companies:

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

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

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

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

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

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

# Preface

This IBM Redbook will help you develop PHP applications in IBM® database servers, DB2® data server, Informix® IDS, and Cloudscape™. This book is organized as follows.

- ▶ **Chapter 1** discusses the software and concepts commonly used to develop dynamic Web applications with PHP and IBM data servers.
- ▶ **Chapter 2** outlines the requirements and design of the sample application. It introduces the data model and code used for the application and illustrates its basic implementation.
- ▶ **Chapter 3** provides the installation and configuration information for Zend Core for IBM on Linux® and Windows® and Zend Studio.
- ▶ **Chapter 4** discusses installing, configuring, and developing applications using Apache, PHP, and DB2 database. We provide PHP application examples with various DB2 interfaces.
- ▶ **Chapter 5** discusses in-depth the installation and configuration of the application development environment with Informix IDS as the database server. It reviews the PHP extensions available for developing PHP applications with Informix IDS, and shows the functional strength of each extension.
- ▶ **Chapter 6** provides the steps and examples for converting PHP applications from MySQL V5 to DB2 UDB V8.2. It also covers the conversion of database objects' stored procedures, triggers, views, and user defined functions.
- ▶ **Appendix A** gives an overview of SDOs and the motivations for using them in the PHP environment. A simple contact management scenario is used to illustrate key concepts.

## The team that wrote this redbook

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



*Figure 1 From left to right: Daniel, Kiran, Holger, and Piotr*

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

**Holger Kirstein** is a resolution team engineer with the European Informix support team. He joined the Informix support team in 1996 and has over 15 years experience in application development and support for Informix database servers and Informix clients. He holds a Masters of Applied Computer Science from Technische Universität, Dresden.

**Daniel Krook** is an IT Specialist at IBM in White Plains, New York. He has over 10 years experience in Web site development and currently builds applications on WebSphere® and DB2 along with Apache, MySQL, and PHP. He received a Bachelor of Arts in Political Science and International Studies from Trinity College in Hartford, Connecticut. He holds certifications in PHP (Zend Certified Engineer), Java™ (SCJP), and Solaris™ (SCSA). He occasionally writes PHP-related articles for IBM developerWorks®.



**Kiran H. Nair** is Advisory Software Specialist, IBM India Software Labs, working as a consultant in the Lab services team for DB2 Information Management software. His areas of expertise involve design, development, and administration of solutions involving DB2 UDB, WebSphere Portal Server, DB2 Content Manager for Multiplatforms, and DB2 Data Warehouse Edition. Prior to joining the Lab services team, he worked along with the DB2 UDB development team for Optimizer Hints, SQL/XML, and XQUERY-related features. He was also involved in writing the application development samples shipped with DB2 UDB.

**Piotr Pietrzak** is a Advisory IT Specialist at the IBM Systems Group in Poland. He has over 10 years of experience in network operating systems, databases, and software development. His areas of expertise include xSeries® servers, Windows, Linux, SQL Server, and directory and cluster services, as well as building the solutions to solve complex problems within large enterprise environments. Piotr holds a Bachelor of Engineering degree specializing in Computer Operating Systems, and has many industry certifications, including MCSE, MCDBA, and MCSA.

## Acknowledgement

The authors would also like to thank the following people for their contributions to this project:

Grant Hutchison  
Dan Scott  
John Paul Parkin  
Peter Kohlmann  
Rav Ahuja  
**IBM Toronto Laboratory**

Graham Charters  
Matthew Peters  
Caroline Maynard  
Anantoju Veera Srinivas

### **Authors of “An introduction to Service Data Objects for PHP”**

Ted J. Wasserman  
Rekha Nair  
Terrie Jacopi  
Kellen F Bombardier  
**Information Management System, Software Group, IBM**

Edgardo G. König  
**Software Sales, Sales and Distribution, IBM**

## Become a published author

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

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

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

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

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

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

[ibm.com/redbooks](http://ibm.com/redbooks)

- Send your comments in an e-mail to:

[redbook@us.ibm.com](mailto:redbook@us.ibm.com)

# Technology overview

As more companies come to rely on the Internet to support their core business processes, there is an increasing demand for Web sites that are better able to serve their customers and more efficiently eke value out of existing data, services, and IT infrastructure.

Increasingly, businesses are choosing PHP and IBM data servers for the low cost, increased reliability, high performance, and speed of development that this platform provides.

With the recent release of DB2 Express-C as a freely available, full-featured database, and an out-of-the-box PHP development and deployment platform in the Zend Core for IBM, this trend will only increase.

This chapter discusses the software and concepts commonly used to develop dynamic Web applications with PHP and IBM data servers. We introduce fundamental Web application architecture, the infrastructure which supports the applications, and software products which fit each of the required roles.

We discuss the following topics in this chapter:

- ▶ Web applications
- ▶ IBM data servers
- ▶ Apache-powered Web servers
- ▶ PHP
- ▶ PHP development tools

## 1.1 Web application environment

As the Web matures, Web sites take on greater responsibility. Once the exclusive realm of scientists posting interlinked documents online, the Web has evolved to support complex business applications and to provide a new interface for traditional IT infrastructure.

The extended capabilities of Web browsers and Web servers enable the transformation of Web sites from simple static file repositories to dynamic applications. These Web applications serve users the way desktop programs have in the past, but through a common interface, and Web applications benefit providers by offering a globally networked user base and simple deployment model.

Gone are the days when it was sufficient to provide the digitized equivalent of printed marketing materials on the Web. As sites become ubiquitous, users increasingly rely on them to serve their shopping, business, and entertainment needs. Users demand information quickly and in a format that's intuitive, stable, fast, and inline with the state-of-the-art in Web development. The less able users are to make use of your site, the more likely they are to use a competitor's. A site must hit the market fast, evolve quickly, and scale to meet the demands of its users. The Web site must be flexible and agile.

At the forefront of these new realities are Web applications powered by PHP, which harness the capability of IBM data servers.

### 1.1.1 Web application advantages

Web applications now serve many of the same needs as traditional applications and their popularity is a result of the following strengths over both desktop applications and other client/server architectures. Web applications are:

- **Protocol-based**

Because communication takes place over a standard protocol, applications are largely platform independent. It does not really matter which operating system or Web browser the user has, as long as it can issue HTTP requests and receive HTTP responses.

- **Programming language independent**

Application developers can write code to specified Web standards instead of Web browser vendor capabilities, which also strengthens platform agnosticism. In fact, users can issue requests from any client that speaks HTTP and provide an arbitrary list of file types HTTP finds acceptable, which HTTP then negotiates with the Web server. For example, a Web browser on a PC desktop might request a Web page and receive an HTML document in

return, whereas a wireless device might receive a WML document via a request to the same URL.

- ▶ **User interface independent**

The user interface is decoupled from the server application. Software can be upgraded without any action required on the part of the user. Similarly, the user can upgrade or change their client, independent of the application on the server.

- ▶ **Scalable and multi-user**

Many concurrent users worldwide can use the application. The application's user base can increase without requiring the active delivery of software by the application developer.

- ▶ **Secure**

Web servers can engage an arbitrary number of other services to assist in request processing, but there only needs to be a single TCP/IP port open for the Web server itself on which to listen.

### 1.1.2 Web application challenges

On the other hand, there are drawbacks to the Web application model which you must mitigate:

- ▶ **User interface**

As a result of decoupling the front end from the back-end, the application designers cannot guarantee that the interface provided to the user is exactly how they intended it to look or indeed how it responds to user interface elements. This means more time testing the application and degrading gracefully in order to not lock out potential users.

- ▶ **Network performance**

The quality of the network and the speed of the connection between the client and the server can limit the performance of the application. Be careful how much information needs to travel back and forth in the HTTP request response cycle.

- ▶ **Data protection and integrity**

Because the application depends on network infrastructure, you must secure any sensitive data transmitted between the user and server. Additionally, since there are many potential users, you must be careful to ensure that each user's data is kept isolated.

### 1.1.3 The state of the Web application world

This is an exciting time to develop Web applications. In the last year or so, the buzzword “Web 2.0” has been used to mark the turning point in the weakness of the traditional model of Web development characterized by the drawbacks above.

Most of the challenges have been mitigated by nearly uniform support for Web standards in the currently popular browsers, and new approaches to the HTTP request and response cycle, such as Ajax, which eliminates the transfer of redundant data and can improve overall networked application performance.

### 1.1.4 Web application components

Web applications are most often comprised of the following three tiers, which this chapter introduces to give you a starting point for exploring in the rest of the book.

We describe each type of software service below in more detail, in the context of Web applications built on an architecture commonly found in IBM environments.

#### ► The presentation tier

This tier serves as the user interface. A Web server fields requests from users and determines from its configuration whether it can service them itself by retrieving a document from the file system or whether it needs the assistance of another program in the logic tier to complete processing.

#### ► The logic tier

The logic engine provides an interface to which the Web server can delegate requests. It performs the business logic of the application. It can accept input, perform operations on it, and return dynamically created output to the Web server to pass back to the client.

#### ► The data tier

A data store manages persisted information and provides an interface to change or retrieve it based on rules and conditions.

Database servers, such as IBM DB2, Informix, and Cloudscape provide the services of the data tier. Apache HTTP Server and its IBM-enhanced offering IBM HTTP Server serve the role of the Web server. And, PHP is used as the business logic engine which most often generates a response, based on information stored in the database, to return to the Web server.

Figure 1-1 on page 5 depicts Web application infrastructure components and the HTTP request-response cycle.

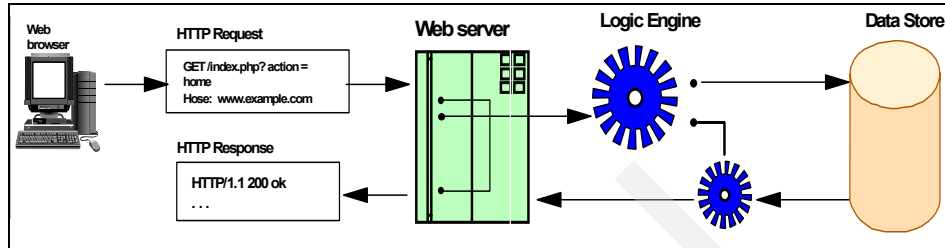


Figure 1-1 Web application infrastructure

## 1.2 IBM data servers

IBM provides robust databases servers for various business needs. In this section, we introduce IBM data server products DB2, Informix, and Cloudscape.

### 1.2.1 DB2 data server

IBM DB2 Universal Database™ (UDB) is an advanced relational database management system that adheres to open standards and is capable of managing both large scale data and high speed transactions. DB2 data server provides a high performance and robust environment for all types and sizes of databases and applications that run on it.

DB2 data server offers database solutions that run on all platforms, including AIX®, Sun™, HP-UX, Linux, Windows, AS/400®, and z/OS® on both 32-bit and 64-bit environments. The DB2 family is a consistent set of relational database management systems (RDBMS) that utilizes shared technologies and a common application programming interface.

The DB2 family includes:

- ▶ **DB2 Universal Database for Linux, UNIX® and Windows:**  
A true cross-platform relational database management system (RDBMS), running on a wide variety of systems, including Windows, Solaris, HP-UX, AIX, and Linux.
- ▶ **DB2 Universal Database for z/OS and OS/390:**  
The premier IBM enterprise RDBMS for use on the mainframe to run powerful enterprise applications, and make large scale e-commerce a reality.

For more details and current information, refer to the Web site:

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

- ▶ **DB2 Universal Database for iSeries™:**  
A 64-bit relational database system that provides leading-edge performance

in e-business and data warehousing environments. The iSeries and DB2 for iSeries in combination provide the flexibility and adaptability to support any type of workload, small or large. iSeries machines are midrange machines running on OS/400® as their operating system.

For more details and current information, refer to the Web site:

<http://www.ibm.com/servers/eserver/iseries/db2/>

► **DB2 Server for VSE and VM:**

A full-function RDBMS that supports production and interactive IBM VM and VSE environments for your company.

For more details and current information, refer to the Web site:

<http://www.ibm.com/software/data/db2/vse-vm/>

► **DB2 Everyplace:**

DB2 relational database and enterprise synchronization architecture for mobile and embedded devices.

For more details and current information, refer to the Web site:

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

## **DB2 for Linux, UNIX, and Windows**

DB2 for Linux, UNIX, and Windows (LUW) is an open, scalable, cost-effective and easy to use database management system from IBM. There are different editions in terms of pricing and features to best suite different types of businesses.

As a distributed database, DB2 provides the following capabilities:

- Integrated support for complex data, such as text documents, images, video, and audio clips
- Integrated Web access through native support for Java, Java Database Connectivity (JDBC™), Perl, PHP, C, COBOL, embedded SQL for Java (SQLJ), and Microsoft® .NET
- Integrated system management tools
- Data replication services
- High availability disaster recovery (HADR)

The following lists the DB2 UDB product offerings for Linux, UNIX, and Microsoft Windows:

► **DB2 UDB Enterprise Server Edition (ESE)**

DB2 UDB ESE is designed to meet the database server needs of mid-sized to large businesses. ESE's high scalability, availability, and reliability features



provide customers an ideal database management system for all types of transactions:

- **The Database Partitioning Feature (DPF)**

The Database Partitioning Feature is a licensing option that allows ESE customers to partition a database within a single system or across a cluster of systems. The DPF capability provides the customer with multiple benefits, including scalability, to support very large databases, or complex workloads, and increased parallelism.

- ▶ **DB2 Express**

DB2 Express features include simplified deployment, autonomic management capabilities, and application development support and design for 24/7 operation. This edition of DB2 UDB is for Independent Software Vendors (ISVs) who want to integrate DB2 UDB into their application at a lower cost and have the capability to expand easily in the future. DB2 Express runs on Windows and Linux. IBM provides 24/7 support for this product.

- ▶ **DB2 Express-C**

DB2 Express for Communities (DB2 Express-C) is a free DB2 UDB version of DB2 Express edition without some extended features. It provides the same core data base server features and development interfaces as DB2 Express edition. You can develop and deploy applications using this database management system. It is easy to upgrade to other DB2 UDB editions, and there is no need to change application code while doing so.

- ▶ **DB2 Workgroup Server Edition (WSE)**

This edition is designed for deployment at a departmental level or in a small business environment with a small number of users. You can deploy WSE on a server with up to four CPUs.

- ▶ **DB2 Workgroup Server Unlimited Edition (WSUE)**

This product offers a simplified per processor licensing for deployment at a departmental level or in a small business environment.

- ▶ **DB2 Personal Edition (PE)**

This edition of DB2 provides a database management system for a single user database.

- ▶ **DB2 UDB Developer's Edition**

This product is for a single application developer to design and build applications.

- ▶ **DB2 UDB Personal Developer's Edition (PDE)**

Similar to the DB2 UDB Developer's Edition, this product enables the developer to build a single user desktop application.

For more information about DB2 offerings, refer to the Web site:

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

### **Enhanced application development productivity**

DB2 UDB offers an extensive toolkit for building applications on both the server and client sides. These application programming interfaces and application development tools focus on maximizing programmer productivity by providing support for major application frameworks, which include Java, C, C++, Perl, PHP, COBOL, and Microsoft.NET programming languages. The capabilities of DB2 for an application developer include:

- ▶ DB2 Development Center for creating server-side objects, such as stored procedures in SQL, Java, or Common Language Runtime (CLR) procedures in C# or .NET languages and user defined functions in SQL, MQ, and XML.
- ▶ Integration in Microsoft Visual Studio® as well as Eclipse-based IBM Rational® Application developer.
- ▶ Enhanced drivers for applications written to .NET, ADO, ODBC, OLE DB, DB2CLI, JDBC, PERL DBD, PHP, and SQLJ programming interfaces.

## **1.2.2 Informix database server family**

The Informix database server family provides a set of server products. Each product has its own strength for the target business environment. The Informix database server family includes:

### **▶ Informix SE**

An easy to embed database server that provides all common database fast and reliable functionality to the clients. There is no ongoing administration effort needed. It is designed for small and medium business environments and is available for Windows, Linux, and UNIX.

### **▶ Informix IDS**

Informix IDS is the database server designed to support all business environments from a small startup company to all worldwide operations in a global account. It provides performance, scalability, and reliability in an object relational database server environment. We discuss the benefits and functionality of the Informix IDS product in a separate section below.

### **▶ Informix XPS**

Informix XPS is intended for use in large data warehouse environments. It provides features to support massive parallel data loads and complex query intensive decision support environments. It extends Informix IDS with the capability to group multiple virtual server instances based on a shared

nothing architecture on one or different machines to one database server for load and resource balancing.

## IBM Informix IDS

Informix IDS is a database server with high reliability and scalability. It combines all the advantages of a multithreaded architecture for effective utilization of the available system resources. It is designed not only for as a mission-critical OLTP server, but also for the DW and BI environments, based on the excellent data fragmentation and task parallelization abilities. With the benefits of an easy-to-use and easy-to-configure server, you can customize Informix IDS for varying business environments from a very small local single user environment up to the high end with thousands of users.

The key features of this database server include:

- ▶ Data and Index fragmentation for load balancing for I/O RW operations.
- ▶ Online backup and restore integrated in the existing backup strategy for operating system objects with a storage manager solution.
- ▶ Fast data load for loading data from external sources or for rebuilding database tables.
- ▶ Parallelization of data read activities (selects, statistic builds, and index builds) for optimizing the available system resource use.
- ▶ Replication (HDR, ER) for providing highest availability using up-to-date standby database servers for switching clients over in case of an outage.
- ▶ User-defined data types to extend the base database types for your own needs (Versions 9 and 10).
- ▶ UDR (User-Defined Routines), written in C, Java (Versions 9 and 10), or stored procedure language.
- ▶ Use of large object data types combined with a appropriate API for developing your own, or reusing already developed business applications named DataBlades (Versions 9 and 10).

Informix IDS is generally available on Windows, AIX, Solaris, TRUE64, IRIX, HP-UX, and Linux. Depending on the base operating system, Informix IDS supports 32 bit and/or 64 bit architecture.

Informix IDS does support many industry standards for connectivity, such as ODBC (CLI), JDBC, and OLE/DB to enable applications to run ANSI SQL and SQL with Informix extensions on the server. Informix IDS also provides its own native interface for implementing fast embedded SQL solutions for specific program languages. This builds a comfortable base for programming languages, such as C, C++, COBOL, PHP, or PERL to integrate the database server in an

application environment. Additionally, it enables you to work with Informix IDS in a VisualBasic, VisualC++, or .NET environment under Windows.

Further information about the Informix database server product family, such as products, availability, features, editions, documentations, and downloads is available under:

<http://www.ibm.com/software/data/informix/ids/>

The IBM Informix IDS V10 database server is available in the following editions:

- ▶ IBM Informix Dynamic Server Express Edition
- ▶ IBM Informix Dynamic Server Workgroup Edition
- ▶ IBM Informix Dynamic Server Enterprise Edition

All bundles contain the Informix IDS database server, the management software for the DataBlades, Informix Connect to provide the connectivity for ODBC and embedded applications, the JDBC driver as the connectivity for Java applications, and the Informix Server Administrator for a Web-based administration interface for the database. The bundles are different in the target business environment.

All editions are delivered as:

- ▶ Informix IDS as the base bundle containing the products listed above
- ▶ Informix IDS with J/Foundation

J/Foundation is an additional product which enables you to build and run your own stored procedures written in Java in the server environment.

### 1.2.3 Cloudscape

Cloudscape is a platform independent, small footprint, standards-based fully functional relational database management system which runs on Java virtual machine (JVM™). Since it runs on JVM, it can run on any platform and any operating system. Cloudscape is powered by Apache Derby engine. In 2004, IBM open-sourced Cloudscape code to the Apache software foundation, which established it as an incubator project under the name *Derby*. After an year in incubation, Derby has now graduated to Apache DB project.

IBM Cloudscape requires no database administration or resource management and is very easy to deploy. Cloudscape has advanced security features and is fine-tuned for high performance and efficient resource utilization. Cloudscape is a multi-user, multi-threaded database which supports advanced database features, such as XA transactions, triggers, constraints, locking, write ahead logging, data caching, group commits, deadlock detection, and isolation levels. The SQL supports stored procedures, views, temporary tables, constraints,

triggers, functions, and joins. Cloudscape also has a very good cost-based optimizer. Cloudscape supports SQL92E standards and partially supports SQL99, SQL/XML, and SQL 2003 standards.

Installing and deploying Cloudscape database is easy since it only involves copying the Derby engine jar files and setting up some environment variables. Cloudscape is designed for zero administration. There is no need for statistics update, space reclamation, and log management. Cloudscape can run in traditional client/server architecture or embedded mode inside a Java application.

In Java applications, you can embed the Cloudscape database engine into the application because the engine includes approximately 2 MB JAR files. The database on disk format is platform independent which makes it easier to export data into another platform. Derby supports major international locales and can be easily localized to run localized data.

Cloudscape powers Zend Core for IBM and WebSphere application server Community edition. It is free to download and use with an option for a fee-based support license with IBM.

Along with the Apache Derby core, Cloudscape provides a set of tools, modules, and packaging. The Cloudscape product contains:

- ▶ Apache Derby core and other tools bundled in an installer package
- ▶ IBM Java Runtime Environment for Linux and Windows
- ▶ IBM DB2 JDBC driver
- ▶ ODBC driver for Cloudscape using DB2 runtime client
- ▶ Database browsing tool called Cloudscape Workbench
- ▶ PDF documentation and information center

### **Cloudscape and DB2**

It is possible to write applications interacting with Cloudscape using C, Perl, PHP, and .NET languages, such as C#.net by using the free downloadable DB2 runtime client. Therefore, you can port all the applications made for Cloudscape to DB2 without any code change.

## **1.3 HTTP Servers**

This section covers the basic functionality that a Web server provides, and an introduction to Apache-powered Web servers, which serve most of the HTTP traffic on the Internet and are most often used in conjunction with PHP.

## Requests and responses

HyperText Transfer Protocol (HTTP) servers, more commonly known as Web servers, are programs which accommodate requests from users of a Web browser, such as Microsoft Internet Explorer or Mozilla Firefox, to provide an interface to resources on the server.

The Web server interprets the user request for a resource identified by its URL, and retrieves either a static document from the file system, or delegates the request to another piece of software configured to act on that type of request.

The other program, PHP, in this example, can in turn request information from the data tier, such as a database running on DB2.

Once the document that PHP prepared is returned, the HTTP server passes it to the client along with status information, or *headers*, indicating whether or not the HTTP server was successful in carrying out the user's request.

## Apache-powered Web servers

The most common Web server used on the Internet is the Apache HTTP Server, which at the time of writing runs on approximately 70% of machines used to handle Web traffic.

Since Apache is open source and distributed under a license which permits redistribution, IBM packages an enhanced version called IBM HTTP Server. This product is preconfigured and optimized to work with IBM middleware and provide additional services commonly demanded by its customers.

The capabilities of both servers are the predominantly the same, and both can support PHP. But there are subtle differences, and you will need to choose between the servers based on how the servers meet your needs. Keep the differences in mind when you develop your applications.

Of course, there are other Web servers available; however, this chapter is limited to discussing Apache and IBM HTTP Server. If you are interested in more options, you can compare features at a Web site, such as:

<http://www.serverwatch.com/stypes/compare/>

### 1.3.1 Apache HTTP Server

Apache has become the most popular HTTP server on the Internet, not only because it is free of cost, but because its modular and highly configurable nature provide excellent flexibility and stability.

You can manage the number of resources the server consumes by running it with a minimum set of features, and extending it as necessary by adding third party or

custom modules. You can also exercise fine-grained control over the number of processes or threads that it uses.

There are two major versions of Apache in wide use on the Internet. While you typically choose the latest version of a piece of software based on its version number, Apache 2 is not simply an upgrade for Apache 1.3, it is a complete rewrite. There are reasons to opt for one version or the other, because of architectural differences resulting from this underlying code change.

## **Apache 1.3**

Apache 1.3 is based on code from one of the first HTTP servers available on the Internet, a program called HTTPd written by Rob McCool of the National Center for Supercomputing Applications (NSCA) at the University of Illinois at Urbana-Champaign in the early 1990s.

For a while, development of the server at the NSCA came to a standstill, so a virtual team of Web site administrators who depended on the server outside of the institution assumed stewardship of the software and improved it by coordinating their own independently developed patches, upgrades, and extensions. Because it resulted from this community-driven collection of individual fixes and improvements, they christened the new release *Apache*, anecdotally, because it was “a patchy server.”

While maintaining the existing code base, the Apache team also began to separately redesign its architecture, and merged their changes with the newly revamped version which provided its characteristic modularity and extensibility. It is this essential branch of code that has remained stable for the last ten years.

It is for this reason that Apache 1.3 is highly regarded as a robust server and at the same time considered to be showing its age.

You can read more details about the evolution of Apache here:

[http://httpd.apache.org/ABOUT\\_APACHE.html](http://httpd.apache.org/ABOUT_APACHE.html)

## **Architecture**

Apache 1.3 runs on UNIX and Linux systems as a collection of processes, which are each allocated their own independent resources by the operating system.

When you start Apache, the first process assumes the role of the parent process, and creates a number of child processes predetermined by its configuration file. The parent process increases or reduces the pool of processes to keep the total within minimum and maximum thresholds configured by the server administrator.

Each of these processes is itself single-threaded, and can manage its resources and loaded modules independently of the others. Any module or extension library

used within an Apache 1.3 server process thus can expect exclusive access to its allocated resources.

To ensure that there are no gradual memory leaks, each process serves only so many requests before it is destroyed, which ensures extreme stability. As a result of this behavior, Apache 1.3 is intended to be very process-oriented.

What makes it so robust on UNIX-like operating systems, however, impacts its performance negatively on non-UNIX computers.

On operating systems such as Microsoft Windows, the process-oriented architecture of Apache 1.3 is emulated and thus does not perform very well because of fundamental differences between how these operating systems manage processes and threads.

In Windows, for example, it is easier to create threads than it is to spawn processes, so the process model does not perform very well. And, the process model does not perform well on an emulation layer which attempts to simulate a UNIX-like interface for the program.

## **Apache 2**

Apache 2 is a complete rewrite of the Web server which contains none of the original HTTPd legacy code, and offers advanced features, such as support for IPv6, stream filtering, integrated SSL support, and new process and threading models which offer better performance on non-UNIX platforms. Apache 2 is also the current focus of maintenance and new features, and it offers better documentation.

Apache 2 supports several Multi-Processing Modules (MPM) which are ways of better using the underlying operating system's native interface instead of emulating another architecture's management of processes and threads. As a result, you can configure Apache 2 to operate most optimally on its host platform.

Choosing a particular MPM among those supported on a UNIX-like operating system can have unintended consequences for certain combinations of PHP and third party extensions that have come to expect a process environment which is not shared by more than one thread.

When running PHP with Apache 2, you should use the “prefork” MPM on UNIX-like operating systems. Though this may decrease performance in certain limited circumstances, it is the current recommendation for running Apache 2 with PHP and it is the default MPM activated on UNIX when building from source.

PHP extensions built for Windows are normally prepared for multi-threaded environments, so it is acceptable to use the default, hybrid multi-process multi-threaded “worker” MPM on Windows servers with PHP in production.



**Important:** Use either the prefork Multi-Processing Module (MPM) for Apache 2 or use Apache 1.3 when selecting a Web server for production use on UNIX-like systems.

The following section of the PHP manual describes the MPM issue in more detail and explicitly states that running Apache 2 in a threaded environment, such as the “worker” MPM, is not recommended for production installations.

<http://www.php.net/manual/en/install.unix.apache2.php>

More information about the MPM options is available in the Apache documentation.

<http://httpd.apache.org/docs/2.0/mpm.html>

## 1.3.2 IBM HTTP Server

IBM provides a pre-packaged Web server based on the Apache code for many platforms as a binary distribution with a point and click installer. It offers this Web server distribution for no cost, although you can get support if you acquired IBM HTTP Server (IHS) along with a license for WebSphere Application Server.

IBM has supported an enhanced version of Apache since 1998. The announcement to base IHS on Apache has advantages for both IBM and the Apache Software Foundation.

The relationship allows IBM to provide customers its software and services on top of a robust, free infrastructure that is at the forefront of Web server advancements.

The Apache Software Foundation gains from contributions by IBM engineers and developers. The following press release outlines the original agreement and motivations.

<http://www.ibm.com/press/us/en/pressrelease/2587.wss>

### What IBM HTTP Server adds

IHS shares many of the core benefits of Apache, but it also has key differences which may influence your decision to use it instead.

#### ► Graphical installer

A platform independent installer provides a common interface to the installation process on all the platforms it supports. This is helpful if you are uncomfortable compiling source code to install Apache on non-Windows platforms.

► **Integrated SSL**

IHS includes a built-in Secure Sockets Layer (SSL) implementation provided by the GSKit library which is certified for use in government applications.

► **Performance**

The IBM HTTP Server includes page cache and CGI improvements for Windows and AIX.

► **LDAP integration**

LDAP integration is included to centrally manage authentication for administration of the Web server. This is helpful for managing a cluster of Web servers.

► **Binary only**

IHS does not ship with source code and thus, you cannot change the MPM, since this choice is made at compilation time. Dynamic shared object (DSO) support to extend the functionality of the server is available, however.

IBM HTTP Server documentation and installation packages are available at the following Web site:

<http://www.ibm.com/software/webservers/htpservers/library/>

You can see the recommended updates here.

<http://www.ibm.com/support/docview.wss?rs=177&uid=swg27005198>

This developerWorks article provides more details on the differences between Apache and the IBM HTTP Server, particularly in regard to PHP.

<http://www.ibm.com/developerworks/opensource/library/os-phphttp/>

**Note:** IBM provides paid support for IBM HTTP Server, but this support does not apply to third party modules such as PHP.

## **IBM HTTP Server 1.3**

At the time of writing, the latest version of IHS 1.3 is 1.3.28 which is based on Apache 1.3.28. It is the only version built with a single process, single thread model.

If you intend to use PHP with IHS on non-Windows platforms in production, you should use this version. You can use other versions, but they are not guaranteed to work exactly as intended in all situations and with all combinations of third party modules, as described earlier.

**Important:** If you are going to use a version of IHS on a UNIX-like server in production with PHP, this is the safest version of IHS to use.

## IBM HTTP Server 6

At the time of writing, the latest version of IHS 6 is 6.0.2.7 which is based on Apache 2.0.47. It is numbered to match the version number of WebSphere Application Server with which it can be managed from the WebSphere 6 administrative console. In contrast to earlier versions, IHS 6 provides documentation in the standard Information Center (InfoCenter) format which is familiar to users of other IBM software.

The “worker” MPM is the default on IHS 6, since this offers the best performance when paired with WebSphere Application Server and IHS is tuned to work best with IBM software. If you intend to integrate PHP with this version of IHS on UNIX-like systems, you should ensure that all of the third party extensions you enable are safe for use with a threaded Web server.

**Note:** If you are going to use a version of IHS on a Windows server™ in production with PHP, this is the best version of IHS to use.

### 1.3.3 Which Web server do I choose?

The following comparison of Apache and IHS characteristics should guide your decision to adopt one or the other. Regardless, both share the same robust infrastructure and are excellent choices for use with PHP.

In this IBM Redbook, we use Apache 2 with the “prefork” MPM on our Linux test environment servers. We note where you should use different configuration options in order to work with Apache 1.3 or IBM HTTP Server.

**Note:** PHP itself is able to run safely on any Apache MPM. Third party extensions compiled with PHP which require non-threadsafe libraries may not be, however. Carefully research any third party modules that you enable for PHP when running in a threaded Web server environment. This will heavily impact which IHS Web server to choose.

## Reasons to choose each Web Server

The following lists reasons to consider one server over another.

### Choose Apache 1.3

- You have a UNIX-like server and are comfortable with a proven, robust server.

- ▶ You do not have or foresee a need for any of the new features that Apache 2 offers.

### **Choose Apache 2**

- ▶ You will run your PHP applications on a UNIX-like system in production and you have no compelling reason to stay with Apache 1.3. Remember to compile Apache with the default “prefork” MPM for this platform.
- ▶ You foresee a need for some of Apache 2’s advanced features.
- ▶ You prefer to work with the currently most supported and documented version and intend to get support from the Web or mailing lists.
- ▶ You will run your application on a Windows or UNIX-like server without WebSphere Application Server.

### **Choose IBM HTTP Server 1.3**

- ▶ You will run IBM middleware and intend to use PHP on a UNIX-like server in production and rely on PHP extensions which use non-threadsafe libraries.
- ▶ The slight performance penalty between this release and later IHS releases is outweighed by the risk of threading conflicts with your third party PHP extensions.
- ▶ You need IBM support (though not available for PHP as a third party extension) or any of the IHS specific features listed above.

### **Choose IBM HTTP Server 6**

- ▶ You rely heavily on IBM middleware in production and you are sure that you are only using thread-safe third-party extensions with PHP.
- ▶ You have IBM middleware and are running on a test or development Unix-like server.
- ▶ You need IBM support (though not available for PHP as a third party extension) or any of the IHS specific features listed above.
- ▶ You are running WebSphere 6 and prefer to manage server operations through the administrative console.

## **1.4 PHP**

The first release of PHP is dated 1994 and was developed by Rasmus Lerdorf under the name PHP/FI (Personal Home Page/Forms Interpreter). Now *PHP* is a recursive acronym to “PHP: Hypertext Preprocessor”.

PHP is a powerful server-side scripting language that was invented and designed for creating dynamic Web applications with non-static content. The PHP code can be a standalone program as well as an insert inside HTML (Hypertext Markup Language) or XHTML (Extensible Hypertext Markup Language). The PHP syntax is based mostly on and similar to C, Java, and Perl. You can use PHP based on an open-source license. You can run the PHP program directly from command line. PHP's modular design also allows you to build an application using the graphical user interface and an extension named PHP-GTK.

Example 1-1 shows a simple PHP program “Hello World!”.

*Example 1-1 Sample PHP program*

---

```
<html>
  <head>
    <title>Example 1</title>
  </head>
  <body>
    <?php
      echo 'Hello World !';
      $HTTP_SERVER_VARS['SERVER_SIGNATURE'];
      echo phpinfo();
    ?>
  </body>
</html>
```

---

In the HTML file, the PHP statements are included inside the `<?php` tag. The `<?php` tag is the signal to the HTML processor that PHP processing is necessary. The `echo` statement is a PHP statement.

The result of this PHP program is similar to what you see in Figure 1-2 on page 20.

System	Windows NT IBM-92D888363F 5.1 build 2600
Build Date	Jan 13 2006 13:49:27
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
PHP API	20020918
PHP Extension	20020429
Zend Extension	20050606
Debug Build	no
Zend Memory Manager	enabled
Thread Safety	enabled
Registered PHP Streams	php, http, ftp, compress.zlib

Figure 1-2 Dynamic Web page generated by the PHP Hello World script

Object-oriented programming (OOP) in PHP is introduced in Version 3 with limited functionality. In PHP 5, many OOP components, such as interfaces, access control, and abstract class have been added. OOP model is mostly based on C++ and Java. Example 1-2 shows a simple OOP program with *interface* and *class*.

#### Example 1-2 OOP PHP program

```
<?php
    interface Book {
        public function author();
        public function writeDescription();
    }

    class Description implements Book {
        private $author;

        public function __construct($author){
            $this->author = $author;
        }

        public function author(){
            return 'Book written by: ' .
                $this->author;
        }

        public function writeDescription(){
```

```
        echo 'Stories for kids mostly';
    }
}

$book = new Description('Hans Ch. Andersen');
echo $book->author();
$book->writeDescription();
?>
```

---

PHP has a modular-based design, but not all the modules are installed by default. For example, separate activation and configuration are required for all database drivers and connectors, including the newly announced PDO (PHP Data Objects) extension. PDO is an interface (some abstraction layer) for accessing databases. Each database driver that implements the PDO interface can expose database-specific features as regular extension functions. Some other official modules that you must activate separately are:

- ▶ Bzip2
- ▶ GTK+
- ▶ Iconv
- ▶ Image
- ▶ IMAP, POP3, and NNTP
- ▶ IRC
- ▶ MCRYPT
- ▶ Ncurses
- ▶ ODBC
- ▶ OpenSSL
- ▶ PDF
- ▶ Service Data Objects
- ▶ SOAP
- ▶ Sockets

The base PHP consists of four main modules. The most interesting modules to programmers are the PEAR (PHP Extension and Application Repository) and PECL (PHP Extension Community Library) repositories. PEAR is a framework and distribution system for reusable PHP components. PECL is an extension which contains many useful free functions based on open source licensing. These modules are created by programmers from around the world.

You can find more detailed information at:

<http://www.php.net/docs.php>

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

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

## Why PHP?

PHP is a popular Web application development language. Here are a few reasons to use PHP:

- ▶ Easy to use:

PHP is a scripting language included directly in HTML. This means that getting started is easy. There is no need to compile PHP programs or spend time learning tools to create PHP. You can simply insert statements and get quick turnaround as you make changes.
- ▶ Fully functional:

The PHP language has built-in functions to access your favorite database. With PHP, your HTML pages can reflect current information from databases. You can use information of the user viewing your HTML Web page to customize the page specifically for that user. You can create classes for object-oriented programming, or use flat file or Lightweight Directory Access Protocol (LDAP) databases. It also includes a spell checker, XML functions, image generation functions, and more.
- ▶ Compatible and quick:

PHP is compatible with all Web browsers, because PHP generates plain HTML.
- ▶ Secure:

Although PHP is open source, it is a secure environment. One of its advantages is that the Web clients can only see the pure HTML code. The logic of the PHP program is never exposed to the client, therefore, reducing security exposures.
- ▶ Open source:

PHP is an open-source programming language. It is easy to get started and find examples from Web sites, such as:

<http://www.sourceforge.net>

## 1.5 Database interfaces with PHP

This section provides a high level overview of the IBM databases that interface with PHP.

### 1.5.1 DB2 and Cloudscape

Both DB2 and Cloudscape can use same PHP APIs to interact with PHP via DB2 Runtime client. The DB2 Runtime client communicates with the DB2 server or



the Cloudscape Network Server using Distributed Relational Database Architecture™ (DRDA®) protocol, which is the industry standard for database communications. All the PHP interfaces communicate to DB2 using Call Level Interface (CLI). The interfaces are PHP extensions, which is part of PECL (PHP Extension Community Library) written in C compiled with DB2 libraries.

There are four main extensions of PHP that you can use to write applications with DB2:

- ▶ IBM\_DB2
- ▶ PDO\_IBM
- ▶ PDO\_ODBC
- ▶ Unified ODBC

IBM recommends using IBM\_DB2 or PDO\_IBM to get the best results out of IBM DB2 database.

## **IBM\_DB2**

The `ibm_db2` extension of PHP provides the interface to connect from PHP to IBM DB2, Cloudscape, and Apache Derby databases. `ibm_db2` provides the mechanism to connect to both cataloged and non-cataloged databases. This extension provides mechanisms for application developers to issue SQL queries, work with large objects, call stored procedures, use persistent connections, and use prepared SQL statements. It also works on PHP releases below Version 5. Unlike `PDO_IBM/PDO_ODBC`, `ibm_db2` is based on traditional procedural programming and performs better compared to Unified ODBC functions. `ibm_db2` provides built-in functions for getting details about the DB2 database server and client by query system catalog tables, which provide lots of information about the DB2 database management system.

## **PDO\_IBM and PDO\_ODBC**

PDO (PHP Data Objects) is an object-oriented, standards-based data access method in PHP, where you can use the same methodology to query the database and fetch data from the supported databases. Both `PDO_IBM` and `PDO_ODBC` extensions are the implementation of PDO specification. The `PDO_IBM` extension provides an IBM database driver for PDO.

When compiled with DB2 libraries, you can use either `PDO_IBM` or `PDO_ODBC` to access DB2, Cloudscape, and Apache Derby databases. These extensions provide a mechanism to connect to both local cataloged and non-cataloged databases. For a local cataloged database, `PDO_IBM/PDO_ODBC` obtain the database server details from the client machine. For a non-local cataloged database, the full details of the remote database are specified in the connection URL. `PDO_IBM/PDO_ODBC` also provide access to advanced features of DB2, such as persistent connections, prepared SQL statements, large objects, and

stored procedures. They provide better performance compared to Unified ODBC functions.

At the time of writing, IBM is also working on supporting the PDO\_IBM driver for Informix Dynamic Server 11.10 and beyond. Customers will have to install the new IBM Data Server Clients to use PDO\_IBM with Informix Dynamic Server. The PDO\_IBM extension utilizes the standard PDO API and is officially supported by IBM.

## Unified ODBC

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

The source code for all the extensions is available for free downloading at the PECL Web site.

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

## 1.5.2 Informix IDS database extensions

Since PHP Version 3, Informix IDS database server serves PHP client applications via the Informix connectivity software (Informix SDK or Informix Connect). PHP applications can connect to the Informix IDS database server using either the native embedded SQL interface or with the invocation of an ODBC library using the CLI (call level interface). The ODBC library can be invoked directly or by using ODBC manager. To use ODBC manager, you must use it in PHP.

PHP provides a huge set of functionality to the application programmer. This functionality is divided into logical groups which we refer to as *extensions*. Extensions have to be included to the PHP base configuration depending on the project needs. You can extend the PHP base with database interface functionality for Informix IDS by adding the extension shared or statically, either in the process of building the PHP from the source or expanding an already existing PHP library with an additional shared library.

There are several database interface extensions. These extensions provide similar base functionality required for database communication in the PHP program with some variations. Choose the extension based on the needs of the project and the functionality provided by the extension. In this section, we provide

a high level overview of the available extensions for Informix IDS. The detailed discussion starting from the setup of all of the components in the development environment to implementation strategies for the Informix IDS database interface in PHP clients is in Chapter 5, “PHP applications with Informix database servers” on page 187.

Currently, you can use three major extensions in PHP to develop an application based on the Informix IDS as the database server.

## **Informix PDO**

Informix PDO provides PHP application development the capability to use an object-oriented database programming interface by classes. Informix PDO provides flexible and fast access to all major database objects, such as tables, views, stored procedures, and database server extensions, such as UDRs. You also can benefit from using the standard database features, such as dynamic statements, all types of cursors, and transactions. Another strength of using the Informix PDO is a unified database application interface supporting a heterogeneous production environment.

Informix PDO is a PECL extension which is based on using ODBC directly for communication with the database server. It provides the Informix database driver for the generic PDO interface, which is generally available in the PHP. This extension has been available since PHP Version 5.1.

In summary, the Informix PDO extension provides the following functionality:

- ▶ Object-oriented programming interface
- ▶ Easily portable
- ▶ Support for static and dynamic SQL
- ▶ Support for BLOB and SBLOB
- ▶ Support for static SQL for complex data types
- ▶ Global support for hold cursor and local support for scroll cursor
- ▶ Database server status and exceptions available
- ▶ Flexible data representation for result sets in the PHP client
- ▶ Support of a built-in transaction interface and the SQL transaction interface
- ▶ Support for stored procedures and UDR with and without return values
- ▶ Based on ODBC (needs Informix Connect for connectivity to the Informix IDS)
- ▶ Extension to the PHP source tree

## **Informix functions (ifx\_\*)**

The informix (ifx\_\*) extension provides a basic procedural interface for developing applications within PHP. It is based on a native (embedded ESQL/C) communication interface with the database. This interface is generally available with the PHP source distribution from Version 3. The ifx\_\* extension provides a

standard set of functions implementing static database interactions. The flexible and comprehensive cursor and BLOB management are its strengths.

The informix extension provides the following functionality:

- ▶ Procedural interface
- ▶ Support for static SQL
- ▶ Support for BLOB into and from different storage methods
- ▶ Database server status for last statement can be monitored
- ▶ Local support for scroll and hold cursor
- ▶ Support for SQL transaction interface
- ▶ Support for stored procedures and UDR with and without return values
- ▶ Based on ESQL/C (needs Informix SDK for connectivity to the Informix IDS)
- ▶ Shipped with the PHP source tree
- ▶ Restricted representation of the result sets in the PHP clients limited to associative arrays

## **unixODBC**

The unixODBC extension is a common interface providing the support for multiple database vendors under the invocation of a third party ODBC manager. This extension is mostly shipped shared with the common Linux distributions, such as SLES9 or RHEL4. The unixODBC provides the application an easy-to-use procedural interface to access the database objects as well as to use the major database features. The database communication is based on the ODBC (CLI) library shipped with the Informix connectivity products. You can use this extension from the PHP Version 3.

In summary, the unixODBC interface provides the following functionality:

- ▶ Procedural interface
- ▶ Easily portable with additional setup for external ODBC manager
- ▶ Static and dynamic SQL support (dynamic SQL is not for queries with result sets)
- ▶ Support for BLOB
- ▶ Global cursor with hold and local scroll cursor support
- ▶ Support of a built-in transaction interface and the SQL transaction interface
- ▶ Database error status for last statement can be monitored
- ▶ Support for stored procedures with and without return values (in dynamic SQL only for stored procedures without return values)

- ▶ Based on ODBC (needs Informix Connect for connectivity to the Informix IDS)
- ▶ Shipped with the PHP source tree

Based on the standard informix extension, there is a PEAR extension available, such as PEAR DB, in order to provide application development the benefit of new object-oriented interfaces or to improve the portability of applications across database installations of different providers.

## 1.6 Zend products

Zend was created by Andi Gutmans and Zeev Suraski in 1997. Zend delivers the premier Web application platform products and services for PHP applications. Zend supports PHP by writing and maintaining the Zend Engine and other PHP extensions such as the SOAP extension. All of these works are open source and donated back to the PHP community. In commercial business, Zend brings products and services to the market to help customers develop, deploy, and manage PHP applications. Zend's solution supports the lifecycle of business-critical PHP applications.

Zend has a fast growing product portfolio, including:

- ▶ Zend Engine
- ▶ Zend Core
- ▶ Zend Platform
- ▶ Zend Studio
- ▶ Zend SafeGuard
- ▶ Zend Optimizer

### ***Zend Engine***

If you are using PHP, you probably also are also using Zend Engine. The Zend Engine is open-source software and available under an Apache-style license. The first version, Zend Engine 1, was introduced in 1999 alongside PHP Version 4. Zend Engine 1 is the core scripting engine of PHP, plus many additional features and platform support beyond Linux/Apache. Zend Engine 2, the current release, provides more revolutionary capabilities including its built-in robust object model.

### ***Zend Core for IBM***

Zend Core is an enhanced version of the open source PHP. Delivering out-of-the-box all the necessary drivers and third party libraries to work with the database, such as DB2, Zend core makes the software installation easier and faster with instance PHP setup. Zend Core for IBM is a fully Zend/IBM certified version of PHP for the IBM databases. The product includes tight integration with IBM database server DB2 and Cloudscape. It also provides native support for

XML and Web Services. It delivers a rapid development and deployment foundation for database driven applications and offers an upgrade path from the easy-to-use, lightweight Cloudscape database to the mission critical DB2, by providing a consistent API between the two.

Product highlights are:

- ▶ PHP 5
- ▶ Easy to install
- ▶ Graphical Web-based Administration Console for Cloudscape servers and PHP environment
- ▶ Bundled IBM Cloudscape server and DB2 drivers for enterprise database Web application deployments
- ▶ Easy to access documentation
- ▶ Support options available from Zend

IBM and Zend are collaborating on development and support for the PHP environment.

This collaboration reinforces the IBM commitment to the open source community and foundations, such as Linux, Apache, Eclipse, and Derby. The IBM newly introduced and optimized DB2 and Cloudscape extension for PHP are submitted back to the PHP community and integrated into Zend Core for IBM.

This expands the IBM investment in open source by helping developers more effectively create and deploy applications. The availability of Zend Core for IBM will significantly enhance developer support for the IBM Cloudscape open-source database, since PHP is one of the most popular Web programming languages in the world. In addition, the Zend Core PHP maintenance process ensures better stability and reliability of PHP as a result of intensive testing and bug fixes that are posted back to the PHP community of users across all facets of the market.

### ***Zend Platform***

Zend Platform makes the life easier mostly for administrators in the enterprise environment. It includes enterprise management and intelligence features, easing the administration of multiple PHP servers. The Zend Central is a central management console for administrators to manage all of their PHP servers from the configuration and performance points of view.

The main features are:

- ▶ Central console:
  - Delivers full exposure of all aspects of your PHP environment
- ▶ Comprehensive application insight

- Outstanding run-time profiling and performance monitoring
- ▶ Audit trail:
  - Error recreation and full problem context
  - Proactive alerts dispatched to relevant IT personnel
  - Online debugging and immediate error fixing, as part of Zend Studio integration
- ▶ Performance boost:
  - Run-time code optimization
  - Code acceleration and precompilation
  - Full caching solution
  - Download optimization and acceleration
  - Full support for PHP 4 and PHP 5
- ▶ Unmatched functionality:
  - PHP session clustering
  - PHP intelligence
  - PHP performance management
  - PHP configuration control
  - PHP/Java integration bridge
- ▶ Session clustering:
  - Performance boost (up to 10x)
  - Locking mechanism to ensure data integrity
  - Works seamlessly with existing PHP code
  - Integrates with load balancers
  - Linear scalability for additional machines

For more detailed information, refer to:

[http://www.zend.com/products/zend\\_platform](http://www.zend.com/products/zend_platform)

### **Zend Studio**

Zend Studio is an Integrated Development Environment (IDE) to provide PHP developers a complete development and testing environment. While working on the PHP script, Zend Studio provides functions for editing, debugging, analysis, and optimization. It also include a set of tools to work with various databases.

Zend Studio features include:

- ▶ Code folding
- ▶ Nested PHP code completion
- ▶ Subversion integration
- ▶ Web services support
- ▶ Integrated debugger
- ▶ Profiler

Zend Studio 5 is available in three editions: Enterprise, Professional, and Standard. For more detailed information, refer to:

[http://www.zend.com/products/zend\\_studio](http://www.zend.com/products/zend_studio)

### ***Zend Safeguard***

Zend Safeguard can help provide software protection. Since PHP is an open source language, PHP application distribution faces many management issues, such as:

- ▶ Anyone can access the source code of the application and change or modify it at anytime.
- ▶ Impossible to protect against copyright infringement.
- ▶ It is almost impossible to implement an efficient licensing model because administrators or programmers can change anything.
- ▶ Impossible to secure internal process business logic.

Zend Safeguard provides a solutions for these problems. Zend Safeguard consists of two product modules that provide software protection:

- ▶ **Zend Encoder**

Zend Encoder compiles and converts plain-text PHP scripts into a platform-independent binary format. Then you can distribute these encoded binary files.

- ▶ **Zend License Manager**

Zend License Manager provides the option to deploy license restrictions to encoded files. You deploy the license file with the encoded PHP script and the license file is validated at run time. License manager allows you to implement your application across your organization, effectively licensing policies on commercial applications. Licensing models are easy to configure and the options are flexible.

### ***Zend Optimizer***

Zend Optimizer is a free downloadable application which allows you to run the code encrypted by Zend Safeguard. Optimizer can also increase PHP run-time performance.



## Sample scenario description

In order to take advantage of the potential of PHP and IBM databases, it is helpful to illustrate some of the features that are available in these growing technologies.

For example, PHP 5 supports much improved object-oriented capabilities, in contrast to prior releases where the data encapsulation implied by objects was not enforced, since access modifiers, such as public or private were not available.

If you are migrating from MySQL, you will encounter robust support for database features that you may not have been able to take advantage of in past versions, such as stored procedures, triggers, views, and user defined functions.

Or if you are Web enabling your existing IBM data server, you will find that you can leverage your data in new ways quickly with the rapid application development cycles that PHP makes possible.

In each case, the sample application will highlight what is now possible. The PHP code shows how enterprise design patterns can be applied in your applications, and the database structure is presented in a way where stored procedures, transactions, UDFs, and triggers ensure proper data handling.

In this chapter, we outline the requirements and design of the sample application which models a Web site used for automobile shopping. It is hereafter referred to as the Dealership.

This chapter introduces the data model and code used for the application and illustrates its basic implementation. The physical definition of the data model is left to the individual product chapters as is the configuration of PHP with the relevant database products.

This chapter covers the following:

- ▶ The usage scenarios that the Dealership Web site must support.
- ▶ The database tables which are required to support the application.
- ▶ The business objects whose representations are operated on by the system.

The Dealership application has been designed with the intention that it should provide a usable implementation of a real-world scenario. Use this to develop your own applications based on best practices for PHP programming and taking advantage of IBM database features.

As such, it provides examples of approaches to development that should be part of any well-designed PHP application. Some of the more important concepts that it illustrates are:

- ▶ How to approach PHP Web application development
- ▶ How to implement a model-view-controller (MVC) architecture in PHP 5
- ▶ How to use design patterns to provide a flexible class model using the improved object-oriented features of PHP 5
- ▶ How to apply best practices in PHP 5, such as code formatting, documentation, object orientation, input filtering, and output escaping

The data model has been designed so that it covers a variety of real-world data management challenges and solutions using most available data types.

The Dealership application is available for download at the IBM Redbook site:

<http://www.redbooks.ibm.com/redpieces/abstracts/sg247218.html>

The download instructions are described in Appendix B, “Additional material” on page 413.

## 2.1 Application requirements

Before we are able to devise a data model and write code for the Dealership Web application, we must understand the needs of the users that the site is intended to serve.

In this chapter, we use the terms Web application, Web site, and tool interchangeably, but you should be aware of the subtle differences as described in Chapter one.

Requirements gathering establishes an understanding between the developers and the users of the system, which covers basic functionality, and it helps us understand where the application might grow, which enables us to plan for likely expansion that should be kept in mind as we design the architecture of the system.

The Dealership application is a Web site that serves customers looking to purchase an automobile and a dealer who manages the stock of vehicles. It uses separate sections composed of different windows to serve the goals of the two distinct types of users.

### Use cases

The functional requirements of the Dealership, or scenarios describing the services it provides for its users, are captured in the form of *use cases*. Use cases document sequences of actions that provide a measurable outcome to an actor who requests them of the system.

The requirements for each of the two portions of the Web application are described below along with a corresponding screen capture of the user interface. They are then followed by the non-functional requirements, or intangible attributes, of the tool which nonetheless impact the usefulness of the system.

Figure 2-1 on page 34 provides a high level overview of the system, documenting the series of related activities that the Customer Portal of the Dealership enables.

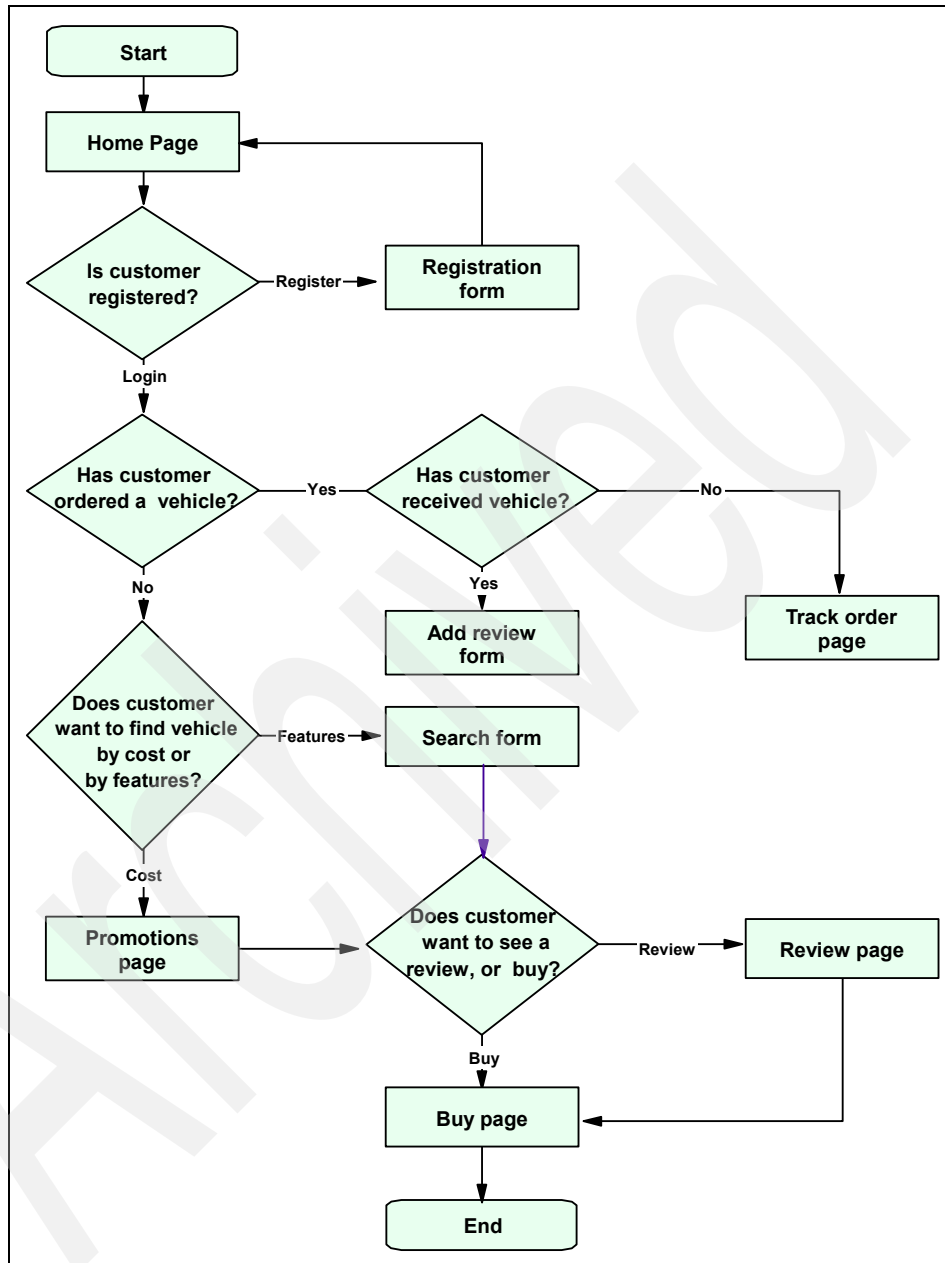


Figure 2-1 Flow chart for Customer Portal of the Dealership application

Figure 2-2 on page 35 provides a high level overview of the system, documenting the series of related activities that the Dealer Portal of the Dealership enables.

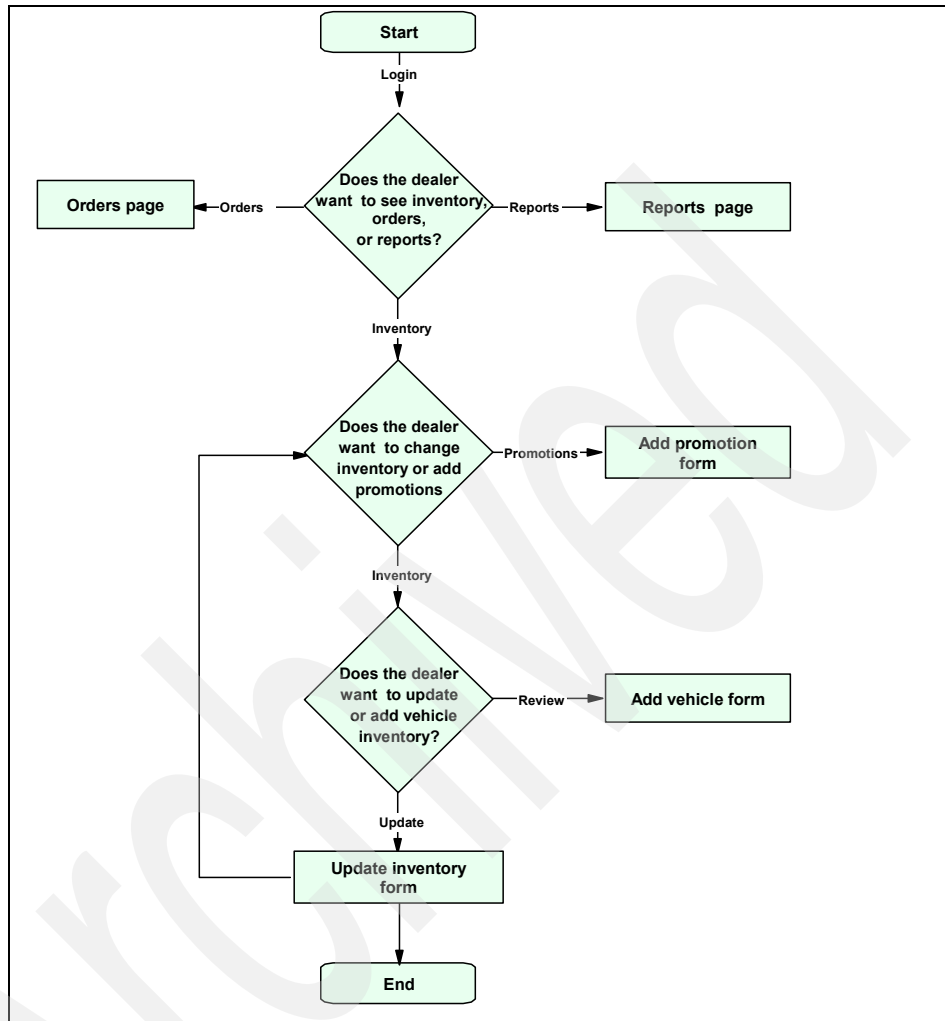


Figure 2-2 Flow diagram for Dealer Portal of the Dealership application

### 2.1.1 Customer interface functional requirements

The customer uses the public portion of the Dealership Web application to find and buy vehicles. To support that end, the following series of activities must be supported.

In this set of use cases, the customer is the lone actor whose actions are performed on the system. The tables list the function overview, preconditions, outcome, and the condition which affects the outcome. For simplicity, most alternate outcomes have been omitted.

Table 2-1 shows the requirements for registering a customer so that the customer can begin to use the Web site.

*Table 2-1 Customer use case: Register*

<b>Overview</b>	The user must register to begin using the Dealership Web application.
<b>Preconditions</b>	None
<b>Outcome</b>	<b>Conditions</b>
The user is registered.	User has provided all information in the correct format and an e-mail address, which is not already in the system.
<b>Description</b>	<b>Basic course of action:</b> <ol style="list-style-type: none"><li>1. The user arrives at the site.</li><li>2. The user provides personal information to register.</li><li>3. The system verifies registration submission.</li><li>4. The system saves the registration information.</li><li>5. The user is granted access to the system.</li><li>6. The use case ends.</li></ol>

Figure 2-3 shows the customer registration interface which requires the user to submit the customer's name and contact details.

DEALERSHIP CUSTOMER PORTAL

For all your vehicle needs

Home

Register

Login

Register

Please enter your information below to register.

Name:

Email:

Street:

City:

State:

--Select an option--

ZIP:

Phone:

Data:

Register

Reset

Copyright © 2006 IBM Corporation :: All rights reserved

Figure 2-3 The customer registration interface

Table 2-2 shows the requirements for customer login functionality. The customer will access the tool by providing the customer’s e-mail address to identify the customer as a registered user of the tool.

Table 2-2 Customer use case: Login

Overview	The user must log in to begin using the Dealership Web application.
----------	---

<b>Preconditions</b>	The user has registered.
<b>Outcome</b>	<b>Conditions</b>
The user is logged in.	User has provided an e-mail address which has been registered.
<b>Description</b>	<b>Basic course of action:</b> <ol style="list-style-type: none"><li>1. The user arrives who wants to use the site.</li><li>2. The user provides the user's e-mail address</li><li>3. The system verifies the submission.</li><li>4. The user is granted access to the system.</li><li>5. The use case ends.</li></ol>

Figure 2-4 shows the login interface which requires the customer to enter the customer's e-mail address.

DEALERSHIP CUSTOMER PORTAL

For all your vehicle needs

[Home](#)  
[Register](#)  
[Login](#)

Login

Please enter your email address below to login.

Email:

Login

Reset

Copyright © 2006 IBM Corporation :: All rights reserved

Figure 2-4 The Login interface

Table 2-3 shows the requirements for searching for a vehicle, which is one of the actions the user can take after the user logs in. In this way, the user can find a vehicle by selecting one or more features.

Table 2-3 Customer use case: Search for vehicle

<b>Overview</b>	The user searches for a vehicle based on desired features.
<b>Preconditions</b>	The user has logged in.
<b>Outcome</b>	<b>Conditions</b>
The user sees zero or more vehicle listings.	The user has submitted search criteria that may not match attributes of vehicles in the system.



<b>Description</b>	<b>Basic course of action:</b> <ol style="list-style-type: none"><li>1. The user provides zero or more vehicle search criteria.</li><li>2. The system verifies the submission.</li><li>3. The system searches for vehicles with matching attributes.</li><li>4. The customer is provided with one or more vehicle listings.</li><li>5. The use case ends.</li></ol>
--------------------	---

Figure 2-5 shows the search vehicles interface which allows the customer to select zero or more features on which to search.

DEALERSHIP CUSTOMER PORTAL

For all your vehicle needs

[Home](#)  
[Search](#)  
[Promotions](#)  
[Track order](#)  
[Add review](#)  
[Logout](#)

Search vehicles

Please select features below to search.

Type:  

--Select an option--

Model:  

--Select an option--

Year:  

--Select an option--

Color:  

--Select an option--

Package:

AC:  

--Select an option--

Power windows:  

--Select an option--

Automatic:  

--Select an option--

Doors:  

--Select an option--

Price:  

--Select an option--

Search

Reset

Copyright © 2006 IBM Corporation :: All rights reserved

Figure 2-5 Search vehicles page

Table 2-4 shows the requirements for viewing active promotions, which is the option the user will choose if the user is more interested in finding a car at the lowest price instead of finding the car by features.

Table 2-4 Customer use case: See promotions

Overview	The user views a list of promotions which apply to individual vehicles.
Preconditions	The user has logged in.
Outcome	Conditions
The user sees a list of promotions.	There are one or more promotions available.
Description	<b>Basic course of action:</b> 1. The user requests a list of promotions. 2. The system searches for vehicles with matching attributes. 3. The user is provided with one or more vehicle listings. 4. The use case ends.

Figure 2-6 shows the promotion list interface which displays all the active vehicle promotions.

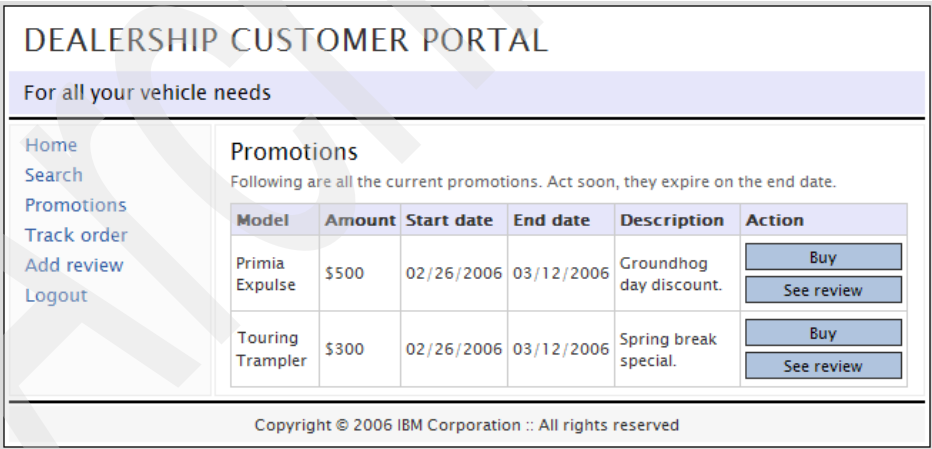


Figure 2-6 Promotions list interface

Table 2-5 shows the See vehicle details user case. After the user has found a vehicle via a promotion or search, the user can view the vehicle details.

Table 2-5 Customer use case: See vehicle details

<b>Overview</b>	The customer views more details on a vehicle.
<b>Preconditions</b>	The user has logged in. The user has a vehicle listing identifier.
<b>Outcome</b>	<b>Conditions</b>
The user sees all the details for a particular vehicle, including reviews.	There is information in the system about the vehicle and reviews from previous customers who have purchased the vehicle.
<b>Description</b>	<b>Basic course of action:</b> 1. The customer requests information on a car. 2. The system searches for information about that vehicle. 3. The customer is provided with vehicle information and reviews. 4. The use case ends.

Figure 2-7 shows the vehicle detail and reviews interface. There are two reviews for the relevant vehicle in the table below the vehicle details.

DEALERSHIP CUSTOMER PORTAL

For all your vehicle needs

Home

Search

Promotions


Track order

Add review

Logout

See reviews

Reviews are below the vehicle description.



Touring Trampler  
2006

Type:	Pick-up	\$27,000
Color:	White	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec risus. Praesent pharetra dictum lectus. Sed eros.
Package:	SL	
AC:	Y	
Power windows:	Y	
Automatic:	N	
Doors:	4	

Customer	Date	Grade	Data
Teemu Lahti	2006-02-26	B	Aliquam purus augue, aliquet ut, facilisis quis, elementum sed, turpis.
Ernesto Guevara	2006-02-26	A	Quisque dictum, tortor in congue feugiat, dolor arcu feugiat risus, nec blandit velit mi vel mi.

Copyright © 2006 IBM Corporation :: All rights reserved

Figure 2-7 The vehicle detail review interface

Chapter 2. Sample scenario description 41

Table 2-6 shows the Order vehicle use case. When the user is ready to buy, the user must have a method for purchasing the vehicle. The table below shows the requirements for placing an order for a vehicle.

Table 2-6 Customer use case: Order vehicle

Overview	The customer places an order for a vehicle.
Preconditions	The user has logged in. The user has a vehicle listing identifier. The user has seen vehicle information and zero or more reviews.
Outcome	Conditions
The user receives confirmation that an order has been placed for a vehicle.	There are one or more vehicles in the dealer's inventory.
Description	<b>Basic course of action:</b> 1. The user orders a vehicle. 2. The system places the order. 3. The customer is provided with order confirmation. 4. The use case ends.

Figure 2-8 shows the order vehicle interface. The user will be prompted to confirm the order.

DEALERSHIP CUSTOMER PORTAL

For all your vehicle needs

Home

Search

Promotions

Track order

Add review

Logout

Buy vehicle

Vehicle information is below.



Touring Trampler  
2006

Type:	Pick-up	\$27,000
Color:	White	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec risus. Praesent pharetra dictum lectus. Sed eros.
Package:	SL	
AC:	Y	
Power windows:	Y	
Automatic:	N	
Doors:	4	

Confirm order

Copyright © 2006 IBM Corporation :: All rights reserved

Figure 2-8 Shows the Buy vehicle interface

Table 2-7 shows the requirements for tracking a vehicle order. After the customer has purchased a vehicle, the user will want to track its delivery details and set its final sales price derived from promotional discounts.

Table 2-7 Customer use case: Track order

Overview	The customer tracks the status of his order.
Preconditions	The user has logged in. The user ordered a vehicle.
Outcome	Conditions
User is able to see order status.	The user's order has been placed and has not been delivered.
Description	<b>Basic course of action:</b> 1. The user requests to track order status. 2. The user is presented with order status. 3. The use case ends.

Figure 2-9 shows the Track order interface with order date, estimated delivery date, and final sales price.



Figure 2-9 The Track order interface

Table 2-8 shows the requirements for adding a vehicle review. After the user has received the vehicle, the user may want to add a review.

Table 2-8 Customer use case: Add review

Overview	The customer adds a review about a vehicle the customer purchased.
----------	--

<b>Preconditions</b>	The user has logged in. The user ordered a vehicle. The user has received a vehicle.
<b>Outcome</b>	<b>Conditions</b>
Customer posts review.	The customer has received a vehicle.
<b>Description</b>	<b>Basic course of action:</b> 1. The user requests to post a review. 2. The system processes and saves the review. 3. The use case ends.

Figure 2-10 shows the Add review form. It allows the user to select one of the vehicles the user owns and the user is able to provide a grade and some comments.

DEALERSHIP CUSTOMER PORTAL

For all your vehicle needs

Home

Search

Promotions

Track order

Add review

Logout

Add review

Please enter information below to add a vehicle review.

Model:

Touring Trampler

Grade:

A

Data:

Add

Reset

Copyright © 2006 IBM Corporation :: All rights reserved

Figure 2-10 The Add review interface

Table 2-9 shows the requirements for logging out of the application when the user has ended the session.

Table 2-9 Log out

<b>Overview</b>	The customer is finished working with the Web application.
<b>Preconditions</b>	The user has logged in.
<b>Outcome</b>	<b>Conditions</b>
Customer is logged out.	The user has confirmed that the user wants to log out.
<b>Description</b>	<b>Basic course of action:</b> <ol style="list-style-type: none"> <li>1. The user requests to log out.</li> <li>2. The system logs the user out.</li> <li>3. The use case ends.</li> </ol>

## 2.1.2 Dealer interface functional requirements

The dealer who owns the Dealership Web application uses a different portion of the site to manage the business operations. To carry out the dealer tasks, the tool must perform the following functions.

In this set of use cases, the dealer is the lone actor whose actions are performed on the system. The tables list the function overview, preconditions, outcome, and the condition which affects the outcome. For simplicity, most alternate outcomes have been omitted.

Table 2-10 shows the requirements for logging in the dealer.

Table 2-10 Login

<b>Overview</b>	The user must log in to begin using the Dealership Web application.
<b>Preconditions</b>	None.
<b>Outcome</b>	<b>Conditions</b>
The user is logged in.	The user has provided an e-mail address which is in the system.
<b>Description</b>	<b>Basic course of action:</b> <ol style="list-style-type: none"> <li>1. The dealer arrives and wants to use the site.</li> <li>2. The dealer provides the dealer's e-mail address.</li> <li>3. The system verifies the submission.</li> <li>4. The dealer is granted access to the system.</li> <li>5. The use case ends.</li> </ol>

Figure 2-11 shows the dealer Login interface where the dealer enters the e-mail address which already exists in the system.

DEALERSHIP DEALER PORTAL

For all your vehicle needs

Home

Login

Login

Please enter your email address below to login.

Email:

Login

Reset

Copyright © 2006 IBM Corporation :: All rights reserved

Figure 2-11 The dealer Login interface

After logging in, the dealer has three options. One of them is to see the dealer's inventory. Table 2-11 shows the requirements for seeing the inventory.

Table 2-11 See inventory

Overview	The dealer wants to see the current state of the dealer's inventory, and optionally update it.
Preconditions	The dealer has logged in.
Outcome	Conditions
Dealer sees inventory.	None
Dealer updates inventory.	The user has updated inventory values for a vehicle.
Description	<b>Basic course of action:</b> 1. The dealer requests inventory status. 2. The system retrieves the current state of the inventory. 3. The use case ends. <b>Alternate course A:</b> 1. Dealer has provided new inventory values. 2. The system processes the inventory values. 3. The use case continues at Step 2 of Basic course of action.

Figure 2-12 on page 47 shows the dealer inventory interface which lists the quantity of each vehicle model available, and provides an option for updating that



number. Promotions may be added to any vehicle which has an inventory value greater than zero. A link is also provided for adding a new vehicle.

DEALERSHIP DEALER PORTAL

For all your vehicle needs

Home

Inventory

Orders

Reports

Logout

Inventory

Inventory information is below. You can also **add** a new vehicle model.

Vehicle model	Quantity	Action
Primia Expulse	<input type="text" value="20"/>	<input type="button" value="Update inventory"/> <input type="button" value="Add promotion"/>
Touring Trampler	<input type="text" value="13"/>	<input type="button" value="Update inventory"/> <input type="button" value="Add promotion"/>

Copyright © 2006 IBM Corporation :: All rights reserved

Figure 2-12 The dealer Inventory interface

Table 2-12 shows the requirements for adding a vehicle. The user may want to add new vehicles to the user’s stock periodically.

Table 2-12 Add vehicle

Overview	The dealer wants to add a vehicle model.
Preconditions	The dealer has logged in.
Outcome	Conditions
Dealer adds a vehicle.	The user has successfully provided information to create a vehicle.
Description	<b>Basic course of action:</b> 1. The dealer requests to add a vehicle. 2. The dealer provides new vehicle information. 3. The system verifies the submission. 4. The dealer is notified that the vehicle has been added. 5. The use case ends.

Figure 2-13 shows the Add vehicle model interface. It contains fields for the vehicle details as well as an area to specify the initial quantity and add a photograph.

DEALERSHIP DEALER PORTAL

For all your vehicle needs

Home

Inventory

Orders

Reports

Logout

Add vehicle model

Please enter vehicle information below to add a new model. You can then update its inventory value.

Type:

--Select an option--

Model:

Year:

--Select an option--

Color:

--Select an option--

Package:

AC:

--Select an option--

Power windows:

--Select an option--

Automatic:

--Select an option--

Doors:

--Select an option--

Price:

Data:

Quantity:

Photo (1.5MB max):

Browse...

Add

Reset

Copyright © 2006 IBM Corporation :: All rights reserved

Figure 2-13 The Add vehicle model interface

Once there are any vehicles added, the dealer may want to provide an incentive to sell them. Therefore, the user must be able to add promotions. Table 2-13 shows the requirements for adding a vehicle promotion.

Table 2-13 Add promotion

Overview	The dealer wants to apply a promotion on a given vehicle.
Preconditions	The dealer has logged in. The dealer has one or more vehicles in inventory.
Outcome	Conditions
Dealer adds a promotion.	The user has successfully provided information to create a promotion.
Description	<b>Basic course of action:</b> <ol style="list-style-type: none"><li>1. The dealer requests to add a promotion on a vehicle.</li><li>2. The dealer provides new promotion information.</li><li>3. The system verifies the submission.</li><li>4. The dealer is notified that the promotion has been added.</li><li>5. The use case ends.</li></ol>

Figure 2-14 shows the add Promotion interface form. It requires a model name, the duration of the promotion, and any caveats or qualifications that should be met.

DEALERSHIP DEALER PORTAL

For all your vehicle needs

Home

Inventory

Orders

Reports

Logout

Promotion

Please enter information below to add a vehicle promotion.

Model:

Primia Expulse

Amount:

Start date:

02/26/2006

End date:

03/12/2006

Description:

Add

Reset

Copyright © 2006 IBM Corporation :: All rights reserved

Figure 2-14 The dealer add Promotion interface

Table 2-14 shows the requirements for seeing orders. Once the user has added vehicles, the user will be interested to see how many vehicles have been ordered and when they should be delivered.

Table 2-14 See orders

Overview	The dealer wants to see the current list of orders.
Preconditions	The dealer has logged in.
Outcome	Conditions
Dealer sees orders.	None

<b>Description</b>	<b>Basic course of action:</b> 1. The dealer requests orders' status. 2. The system retrieves the current state of orders. 3. The use case ends.
--------------------	---

Figure 2-15 shows the Order status page which shows all customer orders by date ordered, delivery date, and the final sales price after discounts.

DEALERSHIP DEALER PORTAL																								
For all your vehicle needs																								
Home Inventory Orders Reports Logout	<b>Orders</b> The vehicle orders submitted by customers are below. They were placed on the in date and should be delivered on the out date. <table><tr><th>Model</th><th>Customer</th><th>In date</th><th>Out date</th><th>Sales price</th></tr><tr><td>Touring Trampler</td><td>Ernesto Guevara</td><td>02/26/2006</td><td>03/12/2006</td><td>\$26,700</td></tr><tr><td>Touring Trampler</td><td>Teemu Lahti</td><td>02/26/2006</td><td>03/12/2006</td><td>\$26,700</td></tr><tr><td>Primia Expulse</td><td>Piotr Akhand</td><td>02/26/2006</td><td>03/12/2006</td><td>\$29,500</td></tr></table>				Model	Customer	In date	Out date	Sales price	Touring Trampler	Ernesto Guevara	02/26/2006	03/12/2006	\$26,700	Touring Trampler	Teemu Lahti	02/26/2006	03/12/2006	\$26,700	Primia Expulse	Piotr Akhand	02/26/2006	03/12/2006	\$29,500
Model	Customer	In date	Out date	Sales price																				
Touring Trampler	Ernesto Guevara	02/26/2006	03/12/2006	\$26,700																				
Touring Trampler	Teemu Lahti	02/26/2006	03/12/2006	\$26,700																				
Primia Expulse	Piotr Akhand	02/26/2006	03/12/2006	\$29,500																				
Copyright © 2006 IBM Corporation :: All rights reserved																								

Figure 2-15 The Orders interface

Table 2-15 shows the requirements for viewing reports. The dealer will want to see trends in how sales are going. This functionality makes that possible.

Table 2-15 View reports

<b>Overview</b>	The dealer wants to see reports on sales.
<b>Preconditions</b>	The dealer has logged in.
<b>Outcome</b>	<b>Conditions</b>
Dealer sees reports.	None
<b>Description</b>	<b>Basic course of action:</b> 1. The dealer requests reports. 2. The system retrieves the current state of reports. 3. The use case ends.

Figure 2-16 shows the Reports interface which shows sales by date and model and determines the average sales price of vehicles sold on that date.

DEALERSHIP DEALER PORTAL															
For all your vehicle needs															
<a href="#">Home</a> <a href="#">Inventory</a> <a href="#">Orders</a> <a href="#">Reports</a> <a href="#">Logout</a>	<b>Reports</b> The following table shows the retail price and average sale price with reports by date. <table> <tr> <th>Model</th><th>Retail price</th><th>Average sales price</th><th>Order date</th></tr> <tr> <td>Touring Trampler</td><td>\$27,000</td><td>\$26,700</td><td>02/26/2006</td></tr> <tr> <td>Primia Expulse</td><td>\$30,000</td><td>\$29,500</td><td>02/26/2006</td></tr> </table>			Model	Retail price	Average sales price	Order date	Touring Trampler	\$27,000	\$26,700	02/26/2006	Primia Expulse	\$30,000	\$29,500	02/26/2006
Model	Retail price	Average sales price	Order date												
Touring Trampler	\$27,000	\$26,700	02/26/2006												
Primia Expulse	\$30,000	\$29,500	02/26/2006												
Copyright © 2006 IBM Corporation :: All rights reserved															

Figure 2-16 The dealer Reports interface

### 2.1.3 Non-functional requirements of the entire application

Non-functional requirements are specifications of the intangible attributes of the application, which nonetheless impact the usefulness of the system. They in turn can affect the implementation of the code and impact architectural decisions just as functional requirements can, so they must be identified before coding begins.

For example, how many simultaneous user requests must the application support? What attributes should it have so that it runs smoothly two years from now?

A good summary of why we must consider these issues in our application architecture early on is provided by Keys Botzum, Kyle Brown, and Geoff Hambrick in the following WebSphere Developer Technical Journal column titled, "Why do non-functional requirements matter?"

[http://www.ibm.com/developerworks/websphere/techjournal/0601\\_co1\\_bobrha/0601\\_co1\\_bobrha.html](http://www.ibm.com/developerworks/websphere/techjournal/0601_co1_bobrha/0601_co1_bobrha.html)

In the case of the Dealership application, there are four non-functional requirements that the client has demanded. The tool must be:

- ▶ Flexible in its interface to the backing data store. A design and implementation must be chosen such that a change in the database product used will not result in a rewrite of business logic. The only change that should be necessary is a configuration parameter change. This would address the customer's migration concerns.
- ▶ Extensible in the ability to add new functionality in the future. It should use object-orientation to represent its domain objects and define interfaces in order to clearly delineate where plug-in points are. It should be evident where new functionality should go.

- Maintainable. The use of Object-Oriented Protocol (OOP) and standard naming conventions allow us to intuitively know where existing functionality resides. It should follow widely accepted coding standards, such as those suggested by the PEAR project:

<http://pear.php.net/manual/en/standards.php>

- Secure. The application must guard against malicious attempts to use the system in ways it is not intended.

## 2.2 Data model

Based on the functional requirements listed above, we can determine the entities and relationships that we must represent in the database.

Figure 2-17 shows a high level version of the data model with columns and keys but no data types. The boxes in the diagram provide a logical description of the tables we have implemented in the sample application. The column types are generic SQL (standard) types. Views, triggers, constraints, user defined functions (UDFs), and stored procedures are not covered, although they are used in the application. For example, a UDF exists for choosing the “vehicle of the week,” and a view is used to make sure that a vehicle addition is done correctly. There is a trigger to decrement the inventory when an order is placed, and so on.

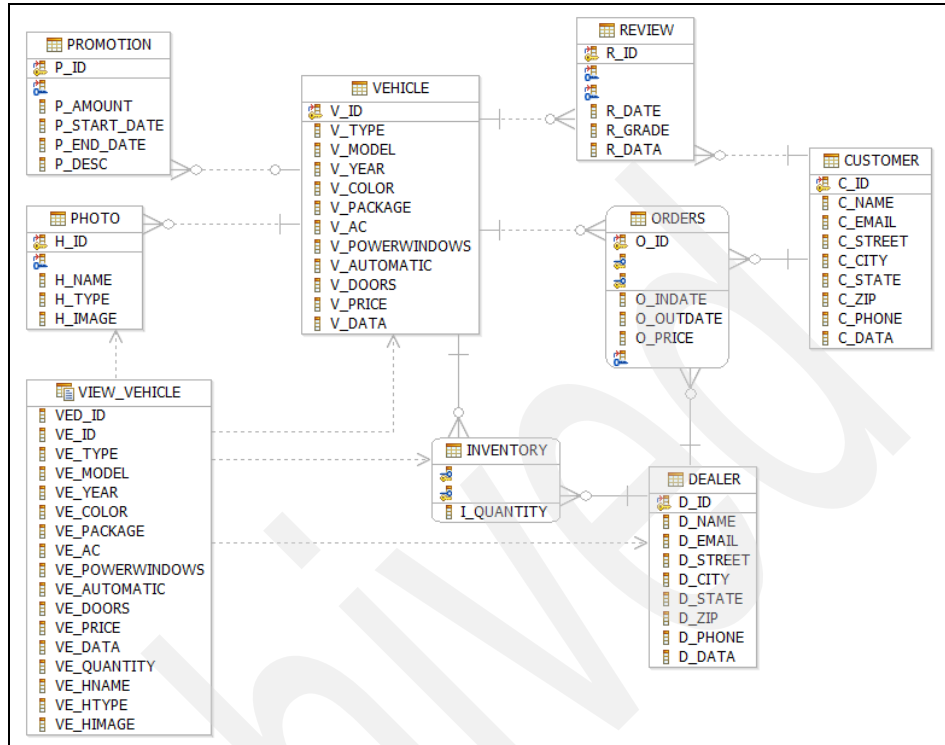


Figure 2-17 The Dealership database entities

## 2.2.1 Tables

The database tables used by the Dealership application are:

### ► CUSTOMER

The CUSTOMER table saves records of users from information they have submitted when they registered. It contains information such as their name, e-mail address, mailing address, phone number, and a brief description. A customer is an independent entity. The primary key is c\_id. There is a unique constraint on c\_email since this is the identifier used to log in.

### ► DEALER

This table contains the single DEALER record. It has the same columns as the CUSTOMER table. A dealer is an independent entity. The primary key is d\_id. There is a unique constraint on d\_email since this is the identifier used to log in.

### ► INVENTORY



The INVENTORY table contains information pertaining to the number of vehicles the dealer has in stock. Each row has a key to a single dealer and single vehicle model along with the relevant quantity. Inventory is a dependent entity which requires a vehicle model and a dealer.

The composite primary key is f\_d\_id and f\_v\_id which are also foreign keys to the DEALER and VEHICLE tables, respectively.

► **ORDERS**

The ORDERS table contains transaction records between a customer and dealer concerning a particular vehicle model. Orders is a dependent entity which requires a customer, dealer, and vehicle.

The primary key is o\_id. The foreign keys are f\_c\_id, f\_d\_id, and f\_v\_id to the CUSTOMER, DEALER and VEHICLE tables, respectively.

**Note:** All database tables are named in the singular, except for the ORDERS table. This is necessary because ORDER is a reserved keyword in SQL.

► **PHOTO**

The PHOTO table contains pictures of individual vehicles. It is dependent on a vehicle. A vehicle may have zero to many photos.

The primary key is h\_id. The foreign key f\_v\_id references the VEHICLE table.

► **PROMOTION**

A promotion is a special discount on a particular model of vehicle good for a certain time period. It is dependent on a vehicle model. A vehicle may have zero to many promotions, which are taken together at purchase time and their value is subtracted from the vehicle price and saved as the sales price in the ORDERS table.

The primary key is p\_id. The foreign key f\_v\_id references the VEHICLE table.

► **REVIEW**

A review is a summary of a vehicle written by a past customer. It is dependent on a customer and a vehicle.

The primary key is r\_id.

► **VEHICLE**

A vehicle is a model of automobile with certain features. This table contains information about a certain class of car. It does not represent an individual vehicle. It is an independent entity.

The primary key is v\_id.

## 2.3 Application design

The PHP code written to support the requirements and data model outlined above is the same regardless of the underlying data store (and indeed the front end layout of the application) because we have chosen a model-view-controller (MVC) architecture.

Though the benefit of PHP is quick turnaround, this can be its downfall, and lead to quickly built but unmaintainable applications. You should try to design up front with the benefit of maintainability, extensibility, and reusability later.

This kind of architecture splits the application into three distinct components with certain responsibilities. This technique *decouples*, or minimizes the dependency between any two components, by establishing three distinct areas of concern:

- ▶ The view

The display of the current application state is handled by HTML templates which contain a small amount of PHP code to display data retrieved from the controller. There is no business logic on these pages, only calls to our model classes to obtain data structures based on user input from the controller. Attributes present in the model or the request delegated by the controller determine whether messages or forms are displayed.

- ▶ The controller

The controller is the single entry point to our application which includes each of our display pages. This thus controls the behavior and data supplied to each of our PHP view pages. The controller also implements security precautions, such as validating and cleansing user input before passing it to the model to be processed.

- ▶ The model

The model consists of our application business logic and objects. This is implemented as a collection of PHP 5 classes. It uses an intermediary data access object which makes calls to a database interface. The database interface translates commands and provides them to the run-time implementation regardless of the actual database or PHP API for communicating with it. Regardless of the concrete database interface, the model objects receive data in the same format and do not need to change if the data tier changes.

Figure 2-18 shows the MVC architecture in the context of the HTTP request-response lifecycle of a user request.

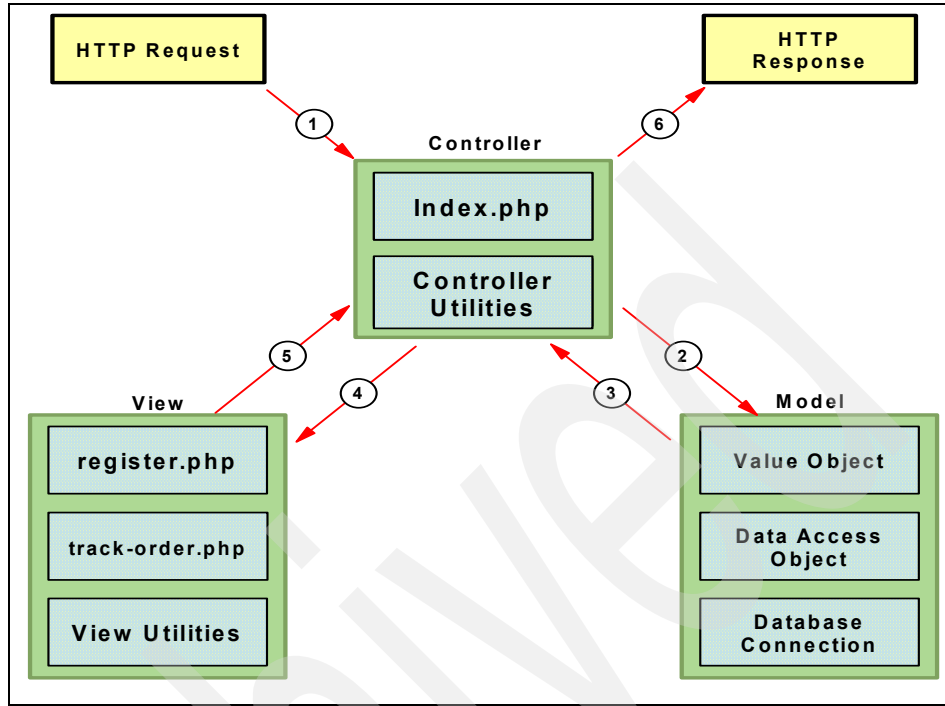


Figure 2-18 MVC architecture

In reality, each of these logical separations bleeds a bit over into the others. For example, in many PHP applications there will be some controller code in the view pages and some model class names mentioned in the display pages, despite a strong MVC approach.

**Tip:** The sample application uses many includes, in order to ease maintenance by breaking functionality into logical units and to best support object-orientation by allocating one file per class. This support for flexibility and maintainability has been weighed against performance issues which may arise from using a large number of includes.

### 2.3.1 Controller components

The controller component is implemented as a single file in both the dealer and customer sections. The file `index.php` is used because it is by configuration (and convention) the default page for which the Web server looks if no file name is given in the URL.

The controller looks for an *action* parameter in the URL, and uses that as a key to look up the corresponding page in a mapping provided by the configuration file for that section, whether customer or dealer portal.

For example, the URL:

<http://localhost/dealer/index.php?action=promotions>

would use the page name specified for the promotions index. In this case, it is `promotions.php` in the dealer folder.

Example 2-1 shows the mapping from `constants.php`.

---

*Example 2-1 Valid customer view page mapping in constants.php*

---

```
// The action mapping used by the customer portal controller.
$validCustomerPages = array(
    'home'      => 'home.php',
    'add-review' => 'add-review.php',
    'buy'       => 'buy.php',
    'login'     => 'login.php',
    'logout'    => 'logout.php',
    'promotions' => 'promotions.php',
    'register'   => 'register.php',
    'results'   => 'results.php',
    'search'    => 'search.php',
    'photo'     => 'photo.php',
    'see-review' => 'see-review.php',
    'track-order' => 'track-order.php'
);
```

---

Example 2-2 shows the logic used for evaluating a user-supplied action parameter, and including the corresponding view.

---

*Example 2-2 index.php determining what view to include*

---

```
<?php
// Check for a valid string to use as a key.
$action = (isset($_GET['action'])) ? $_GET['action'] : '';
$action = ControllerUtilities::sanitizeAction($action);

// If the key exists in the mapping, include the view.
if (!$validCustomerPages[$action]) die('That page does not exist');
include $validCustomerPages[$action];
?>
```

---

Once the view has been included, the controller executes the calls to the model which are provided in the view. See Example 2-3 for an example. Ideally, these PHP view pages should be located outside of the document root so that they

cannot be accessed directly. Or, you should name them with an *.inc* file extension and configure Apache not to serve them.

The following best practices article describes how to achieve this:

<http://www.nyphp.org/phundamentals/sitestructure.php>

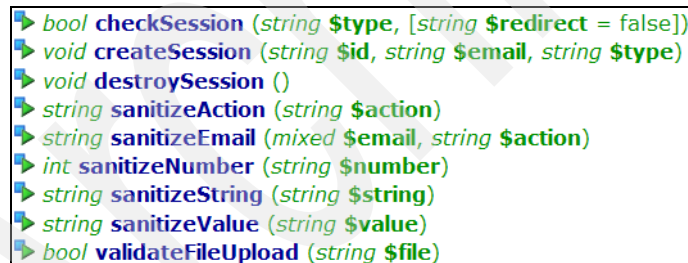
*Example 2-3 Controller processing continues (snippet at top of view action)*

```
<?php
// Check for a valid vehicle id.
$vehicleId = ControllerUtilities::sanitizeNumber($_GET['vehicleId']);

// Find the vehicle.
$vehicleMngr = new VehicleDAO();
$vehicle = $vehicleMngr->selectVehicle($vehicleId);

// Find its reviews.
$reviews = array();
$reviewMngr = new ReviewDAO();
$reviews = $reviewMngr->selectReviews($vehicleId);
?>
```

You may have noticed the code in Example 2-3 uses some static methods. The controller also uses a static utility class to manage sessions, access, and input sanitization. Some of the methods it has available to it are shown in Figure 2-19.



```
bool checkSession (string $type, [string $redirect = false])
void createSession (string $id, string $email, string $type)
void destroySession ()
string sanitizeAction (string $action)
string sanitizeEmail (mixed $email, string $action)
int sanitizeNumber (string $number)
string sanitizeString (string $string)
string sanitizeValue (string $value)
bool validateFileUpload (string $file)
```

Figure 2-19 ControllerUtilities methods

Now that we have a vehicle value object, and a collection of review value objects, it is time to let the view page display them.

## 2.3.2 View pages

We implement each of the windows of the application in HTML pages containing some embedded PHP code which had been included by the main controller script. The view page, which is mapped in the configuration file to an action parameter in the URL, is included by the controller. Continuing the work of the

controller the page retrieves the calls to the model to receive data to display or provides data to the model to process.

Continuing our example from above, here is the display code for the vehicle and reviews. See Example 2-4.

*Example 2-4 Displaying vehicle information obtained by controller from the view*

---

```
Model: <?php echo $vehicle->model ?> <br />
Year:  <?php echo $vehicle->year ?> <br />
Price: <?php echo ViewUtilities::displayCurrency($vehicle->price) ?> <br />
```

---

Example 2-5 shows the code for iterating.

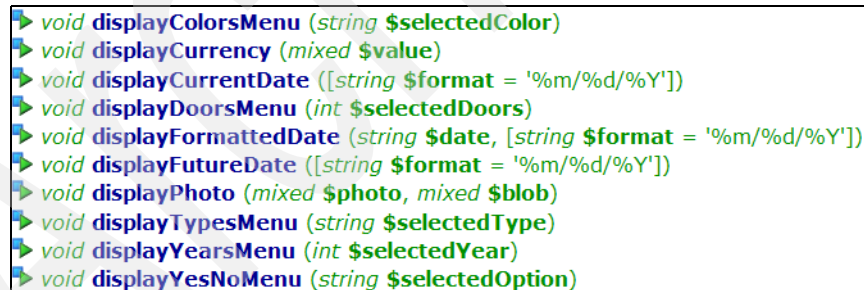
*Example 2-5 Iterating through the reviews for the vehicle*

---

```
<?php foreach ($reviews as $review) { ?>
    Customer: <?php echo $review->customer->name ?> <br />
    Date:     <?php echo ViewUtilities::displayFormattedDate($review->date) ?>
    Grade:    <?php echo $review->grade ?> <br />
    Data:     <?php echo $review->data ?> <br />
<?php } ?>
```

---

The view pages use a static class to facilitate the display of components such as select menus for commonly used options such as a list of years, format currency and dates, and to display binary image data. These methods are shown in Figure 2-20.

A screenshot of a code editor showing a list of static methods in the ViewUtilities class. The methods are listed with their signatures, including parameters and return types. The methods are: displayColorsMenu, displayCurrency, displayCurrentDate, displayDoorsMenu, displayFormattedDate, displayFutureDate, displayPhoto, displayTypesMenu, displayYearsMenu, and displayYesNoMenu. Each method signature is preceded by a small blue icon.

```
void displayColorsMenu (string $selectedColor)
void displayCurrency (mixed $value)
void displayCurrentDate ([string $format = '%m/%d/%Y'])
void displayDoorsMenu (int $selectedDoors)
void displayFormattedDate (string $date, [string $format = '%m/%d/%Y'])
void displayFutureDate ([string $format = '%m/%d/%Y'])
void displayPhoto (mixed $photo, mixed $blob)
void displayTypesMenu (string $selectedType)
void displayYearsMenu (int $selectedYear)
void displayYesNoMenu (string $selectedOption)
```

*Figure 2-20 ViewUtilities methods*

### 2.3.3 Domain objects

The domain objects represent business concepts and map to entities which are implemented as tables in the data model. The business owner speaks of the business processes in terms of these objects, such as Customer or Vehicle, and so we represent these in our data model and application objects.

After we have determined our data model, we can then create objects in PHP to operate on the data via the application. We map classes to tables and columns, and use instances to represent individual rows.

The domain objects fall into two categories.

- ▶ Value objects  
The value objects, also known as *transfer objects* or *beans*, are instances of classes we use to represent entities with properties and a standard interface for interacting with those properties.
- ▶ Data access objects  
Data access objects perform actions with value object instances. This can include issuing requests of the database connection to persist them or returning collections of them.

Keep in mind that this division of responsibilities is only one of many approaches to handling domain objects; however, it is one that provides an appropriate level of conceptual organization for our purposes.

## Value objects

For each table in the database, there is a corresponding PHP class with properties that correspond to its columns. Typically each instance of the class represents a row in the database, and where a foreign key exists, the instance holds a reference to another value object which is an instance of that type.

These types of objects are also known as *transfer objects* and are the primary method of encapsulating information that must travel as a unit among the MVC layers.



Figure 2-21 ValueObject class tree

## ValueObject

*ValueObject* is the parent object of all the value objects. It performs common logic that all value objects need and will inherit. Its methods are shown in

Figure 2-22. The “magic” methods `__get` and `__set` are automatically called when children instance properties are read or written directly. In this way, we can protect our objects from having arbitrary properties set. The `__toString` methods will dump a readable representation when the object is echoed, for example, `echo $vehicle;`

```
+ void __construct ([ValueObject $valueObject = null])
+ mixed __get (string $property)
+ void __set (string $property, string $value)
+ string __toString ()
```

Figure 2-22 Methods implemented by `ValueObject` and inherited by children

The code sample in Example 2-6 shows implementation.

*Example 2-6 Abstract class `ValueObject`*

---

```
abstract class ValueObject {

    protected $fields;

    public function __set($property, $value)
    {
        if (isset($this->fields[$property])) {
            $this->fields[$property] = $value;
        }
    }

    public function __get($property)
    {
        if (isset($this->fields[$property])) {
            return $this->fields[$property];
        } else {
            return false;
        }
    }

    public function __toString()
    {
        print_r($this->fields);
        return '';
    }
}
```

---

Example 2-7 shows an implementation of Value Object. It inherits the magic getters and setters, and so only has to declare its properties in the constructor in the `$fields` variable that it has inherited. See Figure 2-23.



```
 ReviewVO __construct ([mixed $review = null])
```

Figure 2-23 An implementation only must declare its properties in its constructor

Example 2-7 Implementation of a ValueObject by defining properties

```
class ReviewVO extends ValueObject {

    public function __construct($review = null)
    {
        $this->fields = array(
            'id'      => 0,
            'vehicle' => new VehicleVO(),
            'customer' => new CustomerVO(),
            'date'    => '',
            'grade'   => '',
            'data'    => ''
        );
        if (is_array($review)) {
            foreach ($review as $field => $value) {
                $this->$field = $value;
            }
        }
    }
}
```

## Data access objects

For most of the value objects there are corresponding data access objects which manipulate individual instances or collections of instances.

Because the value objects represent rows in the database and are used to hold information instead of action on it, they provide little else than a way to access and change (sometimes called *get and set*) their instance data. We use these helper classes to perform logic on instances and collections of corresponding value objects.

The defined data access objects are as shown in Figure 2-24.

- **DataAccessObject**
  - ◊ InventoryDAO
  - ◊ OrderDAO
  - ◊ PersonDAO
    - CustomerDAO
    - DealerDAO
  - ◊ PromotionDAO
  - ◊ ReviewDAO
  - ◊ VehicleDAO

Figure 2-24 Data access objects

### **DataAccessObject**

*DataAccessObject* (Figure 2-25) is the root class of the data access object and specifies default behavior and required methods. It defines certain methods that all data access objects must support in their implementations and contains a reference to a database connection that its children will use which is determined at run time by the configuration file.

```

+ DataAccessObject __construct ()
+ void deleteObject (string $sql, array $params)
+ void findObject (string $sql, array $params)
+ void insertObject (string $sql, array $params)
+ void performTransaction (array $sql, array $params)
+ void selectObjects (string $sql, array $params)
+ void updateObject (string $sql, array $params)

```

Figure 2-25 Abstract methods of *DataAccessObject*

The class implementation is listed in Example 2-8.

#### *Example 2-8 DataAccessObject*

---

```

abstract class DataAccessObject {

    protected $dbc;

    public function __construct()
    {
        $this->dbc = DatabaseConnection::databaseConnectionFactory();
    }

    public function findObject($sql, $params)
    {
        return $this->dbc->select($sql, $params);
    }
}

```

```

public function insertObject($sql, $params)
{
    return $this->dbc->insert($sql, $params);
}

public function updateObject($sql, $params)
{
    return $this->dbc->update($sql, $params);
}

public function deleteObject($sql, $params)
{
    return $this->dbc->delete($sql, $params);
}

public function selectObjects($sql, $params)
{
    return $this->dbc->select($sql, $params);
}

public function performTransaction($sql, $params)
{
    return $this->dbc->performTransaction($sql, $params);
}
}

```

---

A concrete file, such as VehicleDAO (Figure 2-26), implements these generic methods by determining the appropriate SQL and parameters. This is the interface that the controller works with and it does not need to know about database terminology, such as select, insert, delete, and so on. Example 2-9 shows an extract of the concrete implementation.

```

void deleteReview (mixed $reviewId)
void insertReview (mixed $review)
void selectReviews (mixed $vehicleId)
void updateReview (mixed $review)

```

Figure 2-26 ReviewDAO extends DataAccessObject

Example 2-9 ReviewDAO the implementation of DAO

---

```

class ReviewDAO extends DataAccessObject {

    public function insertReview($review)
    {

```

```

        $values = array(
            $review->vehicle->id,
            $review->customer->id,
            $review->grade,
            $review->data
        );
        return $this->insertObject('
            INSERT INTO REVIEW (
                F_V_ID,
                F_C_ID,
                R_DATE,
                R_GRADE,
                R_DATA
            ) VALUES (
                ?,
                ?,
                CURRENT DATE,
                ?,
                ?
            )',
            $values
        );
    }

    public function selectReviews($vehicleId)
    {
        $rows = $this->selectObjects('
            SELECT *
            FROM REVIEW, CUSTOMER, VEHICLE
            WHERE V_ID = ?
            AND F_V_ID = V_ID
            AND F_C_ID = C_ID
            ', array($vehicleId));
        $reviews = array();
        foreach ($rows as $row) {
            $review = new ReviewVO();
            $review->id = $row['R_ID'];
            $review->vehicle->id = $row['V_ID'];
            $review->vehicle->model = $row['V_MODEL'];
            $review->customer->id = $row['C_ID'];
            $review->customer->name = $row['C_NAME'];
            $review->date = $row['R_DATE'];
            $review->grade = $row['R_GRADE'];
            $review->data = $row['R_DATA'];
            $reviews[] = $review;
        }
        return $reviews;
    }
}

```

}

## 2.3.4 Database adapters

A database adapter communicates with a particular database management system via one of the PHP APIs for database connectivity. Each of these classes implements a common interface to the core Dealership application code via an abstract super object, and are interchangeable as the data store used changes. This is called the *adapter pattern*.

The run-time implementation provided is declared in the configuration file. This is the class tree of the database adapters. See Figure 2-27.

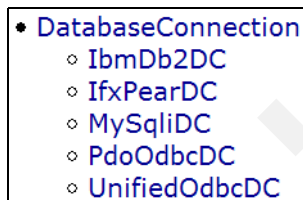


Figure 2-27 Database adapters defined

### DatabaseConnection

This is an interface for issuing SQL commands against the database via one of the physical implementations of the adapters. It contains a static factory method to return a run-time concrete implementation to its caller based on the type specified in the configuration file.

Figure 2-28 shows DatabaseConnection's properties and their data types.

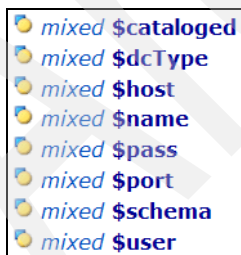


Figure 2-28 DatabaseConnection properties

Figure 2-29 shows DatabaseConnection's methods, parameters, and return types. Note that it does not have a \$db variable. This is a private property implemented in the concrete adapter to provide a reference to the run-time connection to the database provided by the database specific interface functions.

```

void close ()
void connect ()
DatabaseConnection databaseConnectionFactory ()
void delete (string $sql, array $params)
void insert (string $sql, array $params)
void performTransaction (array $sql, array $params)
void select (string $sql, array $params)
void update (string $sql, array $params)

```

Figure 2-29 DatabaseConnection methods

This snippet shown in Example 2-10 shows the code that defines the methods and properties defined by the abstract DatabaseConnection class. The concrete implementation will have to implement the abstract methods and will inherit the protected properties.

Example 2-10 DatabaseConnection's methods and properties

```

abstract class DatabaseConnection {

    protected $host      = DB_HOST;
    protected $port      = DB_PORT;
    protected $name      = DB_NAME;
    protected $user      = DB_USER;
    protected $pass      = DB_PASS;
    protected $schema    = DB_SCHEMA;
    protected $cataloged = DB_CATALOGED;
    protected $dcType    = DB_DC_TYPE;

    public final static function databaseConnectionFactory()
    {
        switch (DB_DC_TYPE) {
            case DB_DC_TYPE_IBM_DB2:          return new IbmDb2DC();
            case DB_DC_TYPE_IFX_PEAR:         return new IfxPearDC();
            case DB_DC_TYPE_DB2_PDO_ODBC:     return new PdoOdbcDC();
            case DB_DC_TYPE_IFX_PDO_ODBC:     return new PdoOdbcDC();
            case DB_DC_TYPE_DB2_UNIFIED_ODBC: return new UnifiedOdbcDC();
            case DB_DC_TYPE_IFX_UNIFIED_ODBC: return new UnifiedOdbcDC();
            case DB_DC_TYPE_MYSQLI:          return new MySQLiDC();
            default:                          return null;
        }
    }

    public abstract function connect();
    public abstract function close();
    public abstract function select($sql, $params);
    public abstract function insert($sql, $params);
    public abstract function update($sql, $params);
}

```

```

    public abstract function delete($sql, $params);
    public abstract function performTransaction($sql, $params);
}

```

---

Figure 2-30 shows the methods that the concrete implementation of a DatabaseConnection must define.

```

➤ void close ()
➤ void connect ()
➤ int delete (array $sql, array $params)
➤ int insert (array $sql, array $params)
➤ array select (array $sql, array $params)
➤ int update (array $sql, array $params)

```

*Figure 2-30 A concrete implementation of a DatabaseConnection methods*

Example 2-11 shows how a concrete adapter which implements DatabaseConnection would look. Notice the private \$db handle which is used to hold a reference to a database connection and which is used to issue queries and updates with SQL.

*Example 2-11 Concrete adapter which illustrates how to plug in new database*

```

class SomeDatabaseApiDC extends DatabaseConnection {

    private $db = null;

    public function connect()
    {
        if (!$this->db) {
            // Database specific API for
            // connecting to the database.
            $this->db = ...;
        }
    }

    public function close()
    {
        if ($this->db) {
            // Database specific API for
            // closing the connection.
            $this->db = null;
        }
    }

    public function select($sql, $params)

```

```

{
    $rows = array();
    $this->connect();
    // Database specific API for querying
    // the database and iterating through
    // the results.
    $this->close();
    return $rows;
}

public function insert($sql, $params)
{
    $this->connect();
    // Database specific API for issuing
    // an update and retrieving status
    // code.
    $this->close();
    return $result;
}

public function update($sql, $params)
{
    return $this->insert($sql, $params);
}

public function delete($sql, $params)
{
    return $this->insert($sql, $params);
}
}

```

---

### ***ibm\_db2 adapter for DB2 and Cloudscape***

The `ibm_db2` database adapter uses the new `ibm_db2` extension as its method of issuing database commands. While `ibm_db2` is a procedural interface, we wrap it in the standard `DatabaseConnection` object-oriented interface. These functions can be used in PHP 4, PHP 5.0, and PHP 5.1, and is the currently preferred method for building new applications on PHP 4 and PHP 5. The detailed discussion and examples are provided in Chapter 4, “PHP application development with DB2” on page 95.

### ***Informix PEAR DB adapter***

The IFX PEAR adaptor uses the `ifx_` extension functions under the hood and a PEAR DB emulation layer using PHP to work around the static SQL limitations of these drivers. This is the preferred connection method in PHP 4 for Informix to



compensate for dynamic SQL features unavailable via the standard `ifx_ODBC` functions. The detailed discussion and examples are provided in Chapter 5, “PHP applications with Informix database servers” on page 187.

### ***PDO adapter for DB2 and Informix***

The PDO database adapter uses the new PDO object-oriented abstraction layer which maps to low level CLI APIs for both Informix and DB2. It uses the `PDO_IBM/PDO_ODBC` driver. It is only available in PHP 5.0 and PHP 5.1. It is similar in concept to PEAR DB as a database independent abstraction standard, but is implemented in C and used as a shared object instead of as a set of PHP scripts; thus, providing better performance over a PEAR abstraction package. The detailed discussion and examples are provided in Chapter 5, “PHP applications with Informix database servers” on page 187.

### ***Unified ODBC adapter for DB2 and Cloudscape***

The Unified ODBC adapter uses the `ODBC_` functions which have been available since PHP 3. This has been the traditional way to support connections to IBM data servers such as DB2 and Cloudscape but has been deprecated in favor of the `ibm_db2` functions for PHP 4 and 5 or the PDO interface for PHP 5. See Chapter 4, “PHP application development with DB2” on page 95.

### ***MySQL adapter***

A MySQL V4.1 and V5 adapter is also provided in the sample code for connecting to more recent MySQL database versions via the `mysqli_` functions.

## **2.3.5 Conclusion**

While PHP and Web applications are only recently gaining prominence as an excellent way to build applications quickly, it should be paired with time-tested software development principles to build flexible and maintainable applications.

A good resource for planning your PHP application architecture is Sherri Wheeler’s “Mature Design Theory in Web Development.”

<http://www.zend.com/php/design/mature-design.php>

A high level, theoretical discussion of application architecture called “What is a software architecture?”, written by Peter Eeles, is available in the Rational Edge:

<http://www.ibm.com/developerworks/rational/library/feb06/eeles/>

## 2.4 Application installation

The Dealership application is available for download at the IBM Redbook Web site:

<http://www.redbooks.ibm.com/redpieces/abstracts/sg247218.html>

The download instructions are described in Appendix B, “Additional material” on page 413.

Following are the Dealership application installation steps:

1. Unzip the downloaded file sg24-7218Sample.zip

The zip file contains the following:

- dlrshp.ddl: sample database DDL
- trigger.txt: trigger DDL
- htdocs: Folder contains all application code.

2. Install Apache 2.0.55 (not 2.2.0). The Apache HTTP Server can be downloaded from:

<http://httpd.apache.org/download.cgi>

3. Install Zend Core and DB2 Express-C via the Zend installer. Remember what you set the passwords to for the database for step 6.

[http://www.zend.com/products/zend\\_core/zend\\_core\\_for\\_ibm](http://www.zend.com/products/zend_core/zend_core_for_ibm)

4. In Zend Core console, turn on the ibm\_db2 extension and error\_reporting and restart Apache.

<http://localhost/ZendCore/>

To turn on the ibm\_db2 extension, **Configuration** → **PHP** → **Extensions** → **ibm\_db2**, turn on the little switch icon on the right (from red to green).

To turn on error\_reporting, **Configuration** → **PHP** → **Error handling and logging**, set `display_errors = on`.

5. Create the database, run dlrshp.ddl to create the tables, then insert the dealer.

- To create the database, open the DB2 Command Line Processor and run the following command:

```
db2 => db2start
db2 => CREATE DATABASE DLRSHP
db2 => TERMINATE
```

- To create tables using dlrshp.ddl, in the DB2 command window, run the following command:

```
db2 -tf dlrshp.ddl
```

There will be some errors here about dropping tables which do not yet exist. Ignore them.

- Create the trigger. In the DB2 command window, run the following command:

```
db2 -td@ -vf trigger.txt
```

- Insert the dealer, in DB2 CLP, run the following statements:

```
db2 => CONNECT TO DLRSH
```

```
db2 => INSERT INTO DEALER (D_NAME, D_EMAIL, D_STREET, D_CITY, D_STATE,  
D_ZIP) VALUES ('Sam Dealer', 'dealer@example.org', 'New Orchard Rd',  
'Armonk', 'NY', '10604')
```

6. Copy all the files in htdocs folder to htdocs

Windows: C:\Program Files\Apache Group\Apache2\htdocs

Linux: /usr/local/apache2/htdocs

7. Open the configuration file and make sure you have the correct user name and password for the database.

Windows:

C:\Program Files\Apache Group\Apache2\htdocs\inc\config\constants.php

Linux: /usr/local/apache2/htdocs/inc/config/constants.php

8. Log in as the dealer (dealer@example.org) and add some vehicles.

<http://localhost/dealer/?action=home>

9. Register as a customer and begin using the Web site.

<http://localhost/customer/?action=home>



# Zend installation and configuration

This chapter discusses the installation and configuration of Zend Core for IBM on Linux and Windows and Zend Studio. This chapter describes:

- ▶ System requirements
- ▶ Installation and configuration steps
- ▶ Cloudscape introduction
- ▶ Zend Studio installation, configuration, and accessing databases
- ▶ PHP application debugging using Zend debugger

## 3.1 Zend Core for IBM

Zend Core for IBM is an integrated solution specifically designed to help developers deploy database applications and services based on the popular PHP. It is an easy to install and support PHP development and production environment. Zend Core for IBM delivers all the necessary drivers and third party libraries out-of-the-box to work with the DB2 or Cloudscape. Zend Core for IBM is a full Zend/IBM certified version of PHP for the IBM databases.

### 3.1.1 Installation: Linux

Zend Core for IBM comes with Cloudscape. During installation, you have the option to download the DB2 Express. If you have already had the database server installed, you can skip the database installation steps. PHP is bundled in Zend Core for IBM and will be installed automatically when you install Zend Core for IBM. If you have already had PHP installed, Zend Core for IBM installed will comment out the PHP LoadModule directive in `httpd.conf`. Zend Core for IBM will make a backup of the existing `php.ini` and `hpptd.conf`.

To install Zend Core for IBM, follow these steps:

1. Check if the target installed Linux platform is supported:
  - Supported operating systems:
    - LINUX (RHEL 3, 4 and SLES 9) on x86, x86-64 and POWER™
  - Supported databases:
    - DB2
    - Cloudscape
  - Supported Web servers:
    - Apache 1.3.x and 2.0.x
2. Download the Zend Core for IBM.

Zend Core for IBM can be downloaded from:

[http://www.zend.com/products/zend\\_core/zend\\_core\\_for\\_ibm](http://www.zend.com/products/zend_core/zend_core_for_ibm)

If You are planning to install DB2 Express-C, you need to download the DB2 Express-C separately. DB2 Express-C can be obtained from:

<http://www.ibm.com/developerworks/downloads/im/udbexp/>

During the installation of Zend Core for IBM, the installation process can download the current IBM DB2 Express from the Zend Web site if required or the local installation file can be pointed to (selected) if already downloaded. The DB2 Express-C version will be bundled in the next release of Zend Core for IBM.

3. Unzip the downloaded file:

```
cd <downloaded_file_folder>  
tar zxvf ZendCoreForIBM-v1.3.1-Linux-x86.tar.gz
```

4. Begin the installation.

```
cd ZendCoreForIBM-v1.3.1-Linux-x86  
./install.sh
```

Figure 3-1 shows the Zend Core for IBM Installation window.

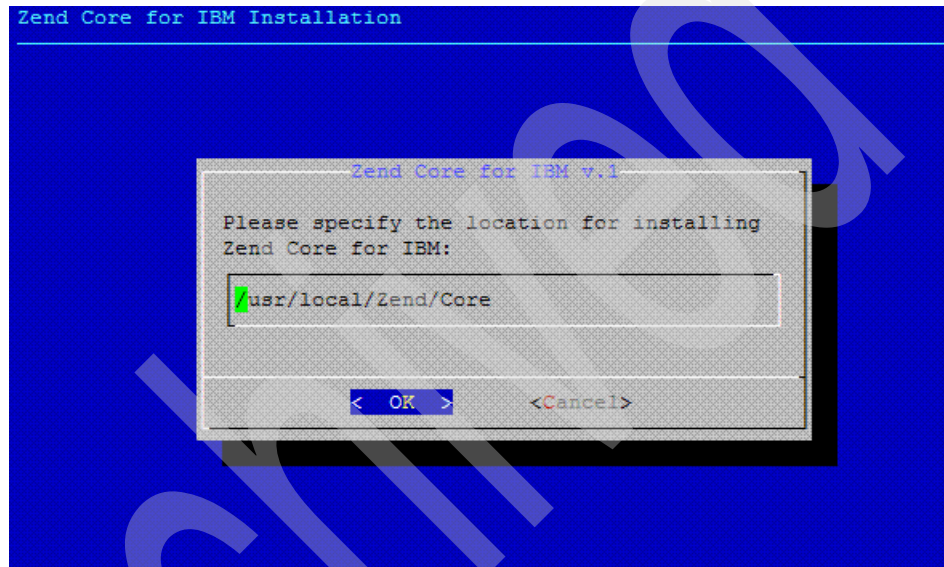


Figure 3-1 Sample installation window of Zend Core for IBM under Linux

## Zend Core for IBM Installation

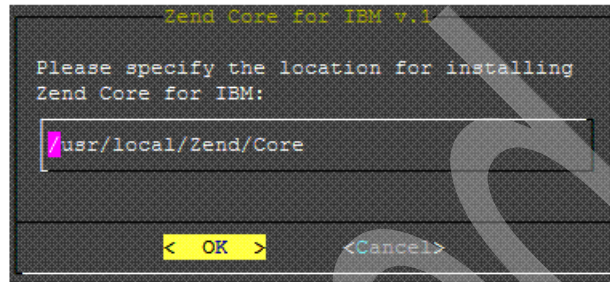


Figure 3-2 Zend Core for iBM Installation

Follow the installation steps from the installer to install the product.

- During the installation process, you will be asked for the Zend Core installation directory, password for administrative purposes, and a few other types of administrative information.
- If you plan to use Cloudscape database, you should select to install the DB2 run-time client. If you plan to install DB2 Enterprise or DB2 Express, do not install the DB2 client because DB2 client will be installed during DB2 installation.
- During the installation of DB2 client, you will be asked for DB2 client instance name, destination folder, instance password, user name, and password for account and group, which must be created for the client inside the operating system.
- At the end of the installation process, you have the option to install Cloudscape.

Zend Core for IBM comes with a sample application called “DB2 Sample Application for PHP”. After the installation process is complete, this sample application is accessible from:

<http://server:port/ZendCore/DB2Sample/>

You can access Zend Core administrative GUI from:



<http://server:port/ZendCore/>

To learn more about DB2 Express or Express-C installation and configuration, refer to this Web site:

<http://publib.boulder.ibm.com/infocenter/db2luw/v8/topic/com.ibm.db2.udb.doc/start/r0008865.htm>

or DB2 documentation:

- ▶ IBM DB2 documentation: *Installation and Configuration Supplement V8*, GC09-4837-01
- ▶ IBM DB2 documentation: *Quick Beginnings for DB2 Clients V8*, GC09-4832-01
- ▶ IBM DB2 documentation: *Quick Beginnings for DB2 Servers V8*, GC09-4836-01

The available Zend Core for IBM documentation is:

- ▶ Installation guide:  
[http://www.zend.com/content/download/1340/7224/version/1/file/Zend\\_Core\\_IBM\\_Installation\\_Guide.pdf](http://www.zend.com/content/download/1340/7224/version/1/file/Zend_Core_IBM_Installation_Guide.pdf)
- ▶ User's guide  
[http://www.zend.com/content/download/1342/7366/version/1/file/Zend\\_Core\\_IBM\\_User\\_Guide.pdf](http://www.zend.com/content/download/1342/7366/version/1/file/Zend_Core_IBM_User_Guide.pdf)

Once Zend Core for IBM is installed, the administrative tool for configuring the environment and changing database settings, instance names, Web server, and so on can be accessed by running the following command:

```
./usr/local/Zend/Core/setup/setup
```

### 3.1.2 Installation: Windows

Before beginning Windows installation, make sure that one of the following supported Web servers is up and running:

- ▶ Apache 1.3.x or 2.0.x
- ▶ Microsoft IIS 5 or 6

And one of the following systems is used:

- ▶ Windows XP Professional
- ▶ Windows 2000 / Windows 2003 Server family

To install Zend Core for IBM on Windows, follow the steps:

1. Download Zend Core for IBM from:

[http://www.zend.com/products/zend\\_core/zend\\_core\\_for\\_ibm](http://www.zend.com/products/zend_core/zend_core_for_ibm)

If you are planning to install DB2 Express-C, download of the binaries is required because they do not come with Zend Core. DB2 Express-C can be obtained from:

<http://www.ibm.com/developerworks/downloads/im/udbexp/>

During the installation of Zend Core for IBM, the installation process will download IBM DB2 Express from the Zend Web site if required. The DB2 Express-C version will be bundled to the next release of Zend Core for IBM.

2. Run the installer and begin the installation process:

ZendCoreForIBM-v1.3.1-Windows-x86.exe

The installation process is similar to that described in 3.1.1, “Installation: Linux” on page 76. The major difference is that Windows has a graphical-based installer. For more information about DB2 installation, refer to:

<http://publib.boulder.ibm.com/infocenter/db2luw/v8/topic/com.ibm.db2.udb.doc/st art/r0006867.htm>

During installation, PHP support is installed on the recognized Web server and the default Web site: <http://server:port/ZendCore/> is created. The sample application is shipped as well. The application can be accessed at: <http://server:port/ZendCore/DB2Sample/>

Figure 3-3 shows the default installation Web site in Microsoft IIS Management console.

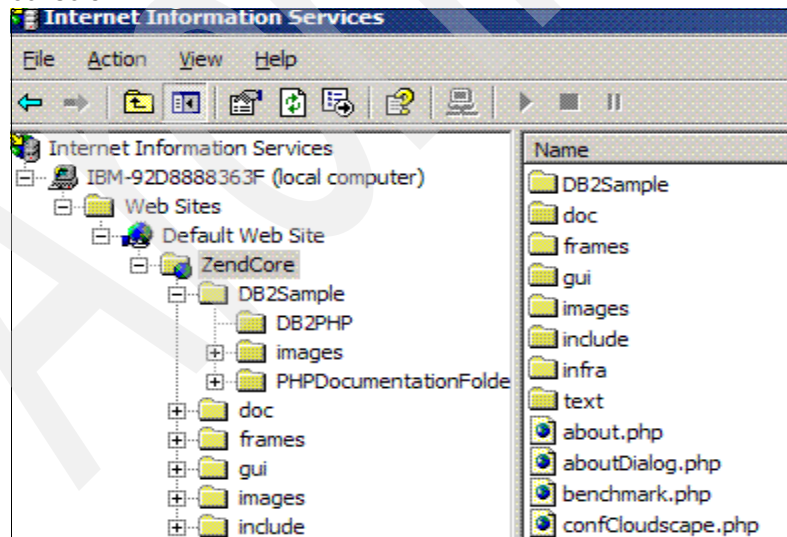


Figure 3-3 Default Zend Core Web site in IIS Manager window

The Zend Core for IBM on Windows environment does not differ from Zend Core for IBM on the Linux platform. GUI and all the available options look exactly the same. A sample GUI window of Zend Core can be seen on Figure 3-4.

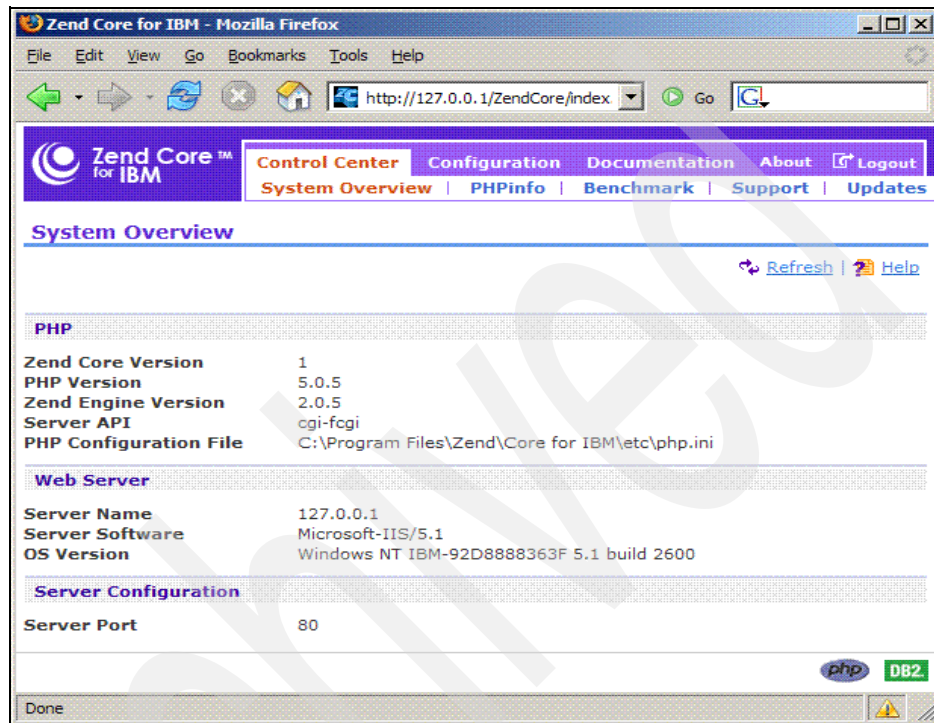


Figure 3-4 Zend Core for IBM GUI installed on Windows platform

For further administration, use the Zend Core for IBM Setup tool. The Setup Tool for Windows is accessed from the start menu: **Zend Core for IBM** → **Zend Core Setup**.

### 3.1.3 Sample application

After the Zend Core for IBM installation finishes, the sample application can be accessed by: <http://server:port/ZendCore/DB2Sample/>. The purpose of this application is to show and describe how to connect to and set up the DB2 (or Cloudscape) database, as well as connect to it from PHP.

The DB2 Sample Application for PHP provided with Zend Core is intended to be more than a selection of sample code. Because it was designed for use with a real application, is well-architected and secured. It is also easily extensible and usable by several developers at the same time. That application sample will save

a great amount of time and help you to understand not just how to build a DB2 PHP application, but also how to write an well-designed one.

The DB2 Sample Application for PHP illustrates how easy it is to write basic PHP applications that can access a DB2 database. It includes examples of how to:

- ▶ Create attractive table displays.
- ▶ Sort by column title.
- ▶ Create drill-down selections.
- ▶ Link to related tables.
- ▶ Create, update, and search for data.
- ▶ Create a history bar.
- ▶ Divide a result page into multiple pages for easy navigation.
- ▶ Ensure security through DB2 authentication and authorization.
- ▶ Store user names and passwords safely using session variables.

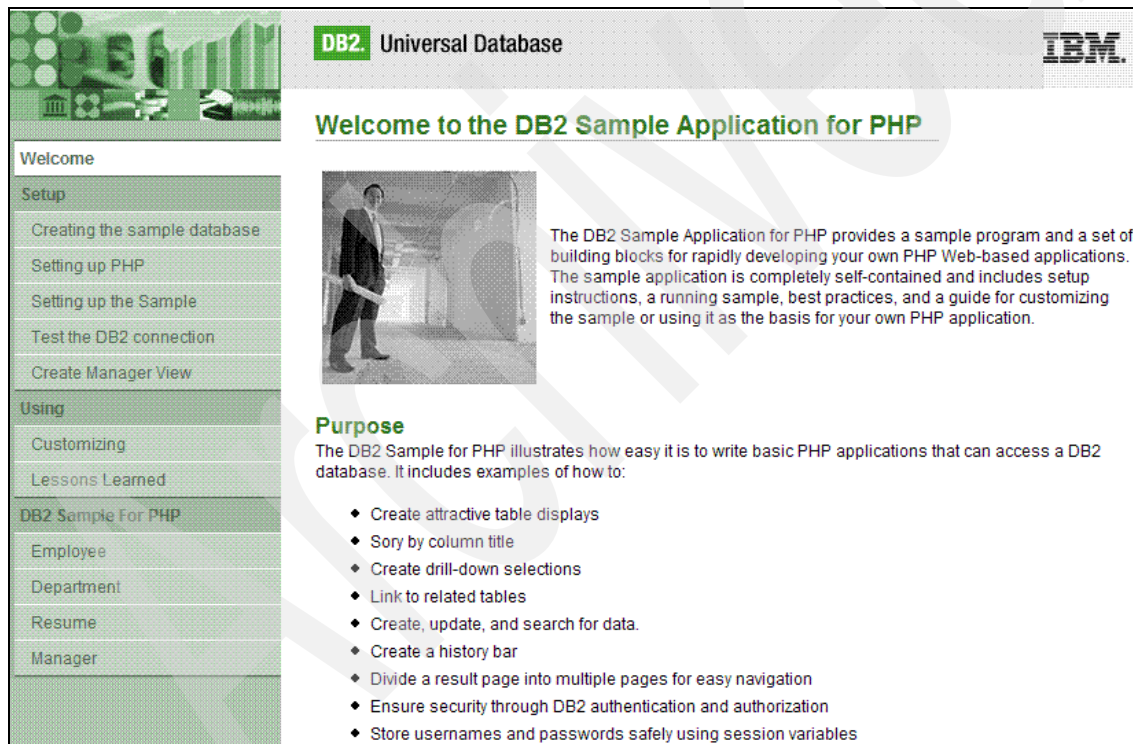


Figure 3-5 Sample application main window

The sample application contains a set of building blocks for rapidly developing your own PHP Web-based applications. The sample application is completely self-contained and includes setup instructions, a running sample, best practices,

and a guide for customizing the sample or using it as the basis for your own PHP application creation.

**Tip:** In the sample application, `db2_connect` method is used to connect to the database. This method is natively supported by Zend Core because of DB2 native client installation. More information about DB2 database connection methods can be found at “Connect to the database” on page 114.

## 3.2 Zend Studio

Zend Studio is an Integrated Development Environment (IDE) for PHP application development. It provides a complete development environment, including editing, debugging, analysis, optimization, and database tools to facilitate the development cycle.

Zend Studio consists of three editions, Zend Studio Standard, Zend Studio Professional, and Zend Studio Enterprise. Zend Studio Professional 5.1 is used in this redbook Lab environment.

### 3.2.1 Installation

An evaluation version of Zend Studio can be obtained from:

[http://www.zend.com/free\\_download/studio](http://www.zend.com/free_download/studio)

You must be a registered user on <http://www.zend.com> to download the product.

**Tip:** The **Zend Core for IBM** direct link is:

<http://downloads.zend.com/studio/5.1.0/>

Once Zend Studio is downloaded, the Web server and PHP version must be considered. General Zend Studio V5.x system requirements are as follows:

- ▶ Zend Studio client - System requirements:
  - Supported platforms:
    - Windows x86 2000, Windows XP, and Windows 2003
    - Linux x86
    - Linux x86-64
    - Mac OS X Power 10.4
  - Zend Studio client - PHP compatibility:
    - Supports all PHP versions

On both Windows and Linux platforms, Zend Studio installation provides an option to download and install all necessary components, including PHP and Apache.

**Note:** Zend Studio Server includes Apache Version 1.3.x and PHP 4.4.x.

- ▶ Zend Studio Server - System requirements:
  - Supported platforms:
    - Linux x86
    - Linux x86-64
    - Solaris Sparc 8, 9, 10
    - FreeBSD x86 5.4, 5.5
    - Windows x86 2000, Windows XP, and Windows 2003
    - Mac OS X Power
  - Supported Web servers:
    - Apache 1.3.x
    - Apache 2.0.x (Prefork mode only)
    - IIS 5, 6
  - Supported PHP versions:
    - 4.2.x up to 4.4.x.
    - 5.0.x, 5.1.x.

In the case that IIS is chosen, refer to Microsoft documentation for how to install the IIS server on a Windows machine. The PHP version for the IIS server can be obtained from:

<http://www.php.net/downloads.php>

The PHP package will configure IIS for you.

Zend Studio V5.1 for Linux and Windows platform has a graphical-based installer and on both platforms installation looks the same. To install Zend Studio V5.1:

- ▶ Linux (remember to be in GUI mode)

```
cd <download_directory>
tar zxvf ZendStudio-5_1_0.tar.gz
./ZendStudio-5_1_0.bin
```
- ▶ Windows

```
<download_directory> ZendStudio-5_1_0.tar.gz
ZendStudio-5_1_0a.exe
```

Figure 3-6 shows one of the installation windows.

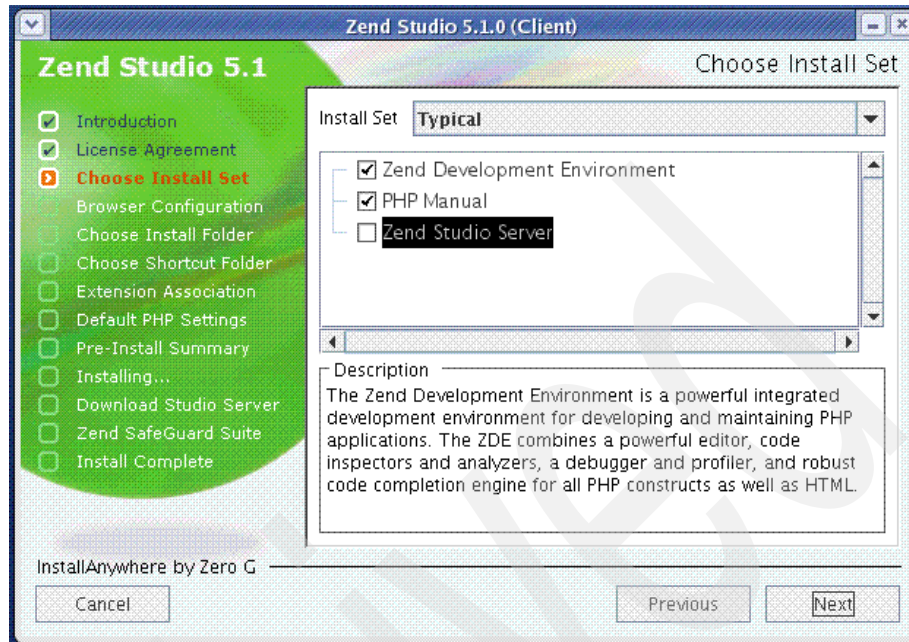


Figure 3-6 Zend Studio 5.1.0 installation window

### 3.2.2 Configuration

Once the installation is complete, you may want to configure the Zend Studio to best suit your development needs. To develop a database Web application, you may want to consider configuring the following items:

- ▶ Integrated Development Environment preferences

To customize Zend Studio, use **Tools** → **Preference**. See Figure 3-7. Go through each tab and set your preference including:

- Debug mode and options
- Source control behavior
- Code completion rules
- File encoding



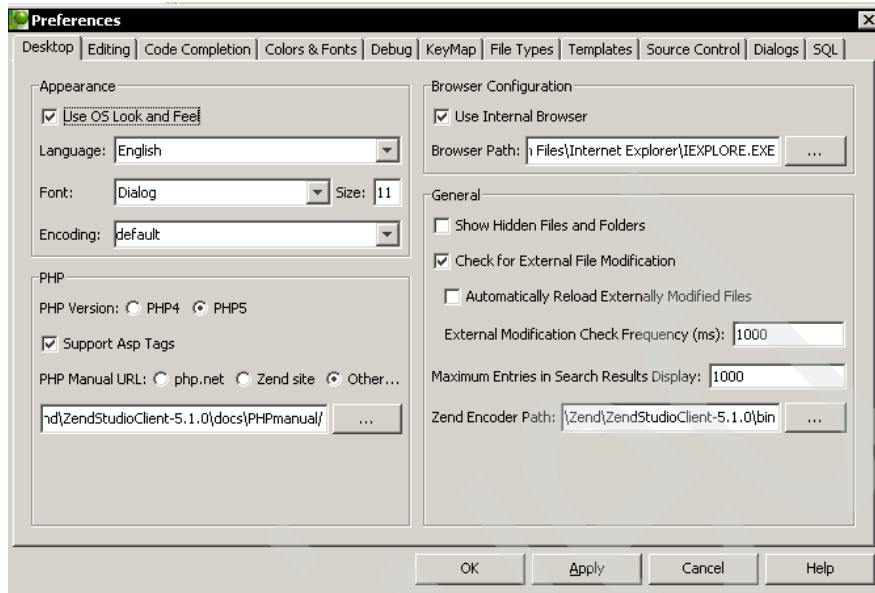


Figure 3-7 Set the preferences

#### ► Project Properties

Projects are a collection of applicator files, folders, and definitions. You can set project folder location, debug mode, debug client port, and more in Project Properties. See Figure 3-8. The project location can be:

- Local or network drives
- FTP sites
- CVS server repositories

When you start a new project, the Project Properties window will be shown for you to override the default settings.

To go to Project Properties, select **Project** → **Project Properties**



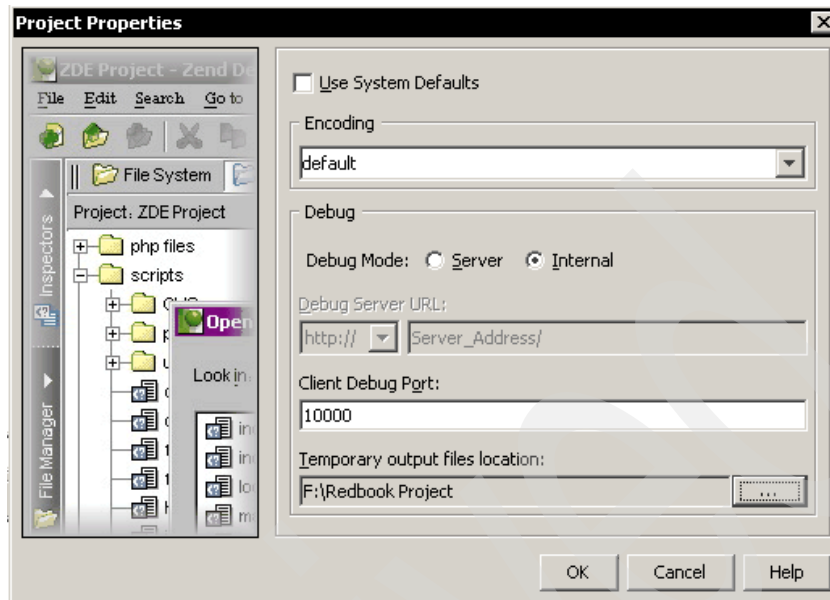


Figure 3-8 Project Properties

► Database connectivity

Zend Studio has a built-in interface which can create a link between the Studio client's development environment and SQL databases. The following SQL servers are supported:

- Cloudscape
- DB2
- MSSQL
- MySQL
- Oracle
- PostgreSQL
- SQLite

The SQL connectivity tool allows you to add a connection to a database server. Once the connection is established, you can work on viewing the database structure, editing the table, and running an SQL statement from the Studio Client development environment.

Zend Studio provides the following SQL support:

- SQL Server Configuration: Allows you to configure the SQL Server Settings
- SQL Server Tree: Allows you to view the database structure that is composed with schema, tables, stored procedures, indexes, and more

- SQL Query Functions: Permit you to run SQL query (statements) on a selected location
- Messages: Return an error message if the query script fails to execute properly
- Results (Edit mode): Allow you to edit the contents of the table.

Text Viewer, Hex Viewer, and Image Viewer™ allow you to view the contents of a table cell that contains textual data or binary data in a form of BLOBs and CLOBs.

To create connection to the required database, click the SQL icon as shown in Figure 3-9.

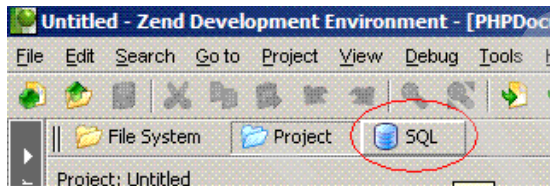


Figure 3-9 SQL server icon

Click the Add icon as shown in Figure 3-10.

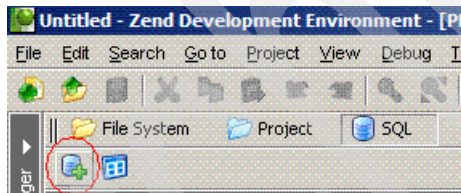


Figure 3-10 Add connection icon

In Add SQL Server panel (Figure 3-11), select the Server Type, enter Server Name, Host Name/IP, DataBase Name, User Name, and Password. Use the Test button to test the database connection.

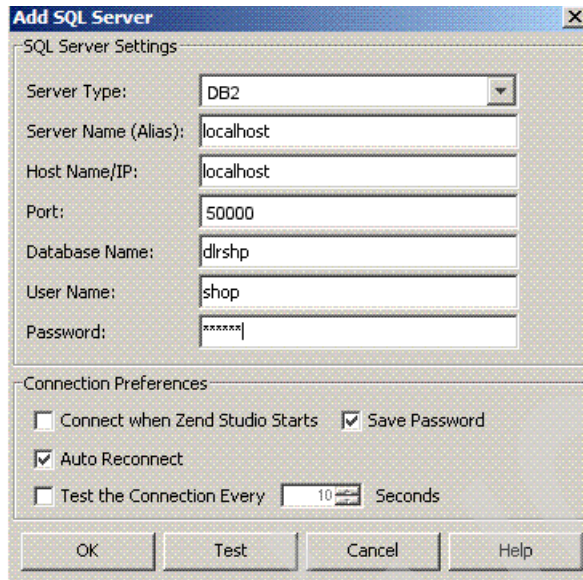


Figure 3-11 Adding SQL server to the Zend Studio client

### 3.2.3 Debugger

Zend Studio has a powerful integrated debugger. You can use Zend Studio to test the scripts remotely or locally. The variables can be changed at run-time during the test. Debugger can be run from the development environment as well as from the Web browser via an integrated panel (Internet Explorer, Mozilla, or FireFox).

Using Zend Studio, two debugger modes can be chosen:

- Internal Debugger

The Internal Debugger enables developers to locally debug developed code before deploying it to a production or test server. The internal option means that only files located in local directories can be debugged. When debugging internal files, the Zend Studio Internal Debugger uses its own PHP version that was installed together with Zend Studio in the installation process. This PHP version is compatible for PHP4 and PHP 5.

**Tip:** Always try to use the same versions of Web server and PHP on the production server to avoid any problems related to version difference.

The following are features that can be used in the debugging process:

- Debug Messages Window: Locate and define errors using the messages generated and displayed in the Debug Messages window.
  - Variable Window and Watches Window: Watch and reference variables, functions, classes, and expressions.
  - Stack Window: Monitor the call stack and passed variables.
  - Debug both the Calling and Called Functions: Using Step in, Step over, and Breakpoints.
  - Control the Debugging Session: Use complete, or line-by-line debugging options using tools such as Breakpoints and Go to Cursor.
  - View and Render Standard Output: Using the content-generated output window.
  - View Buffer: Using the content buffered in the Buffer Window.
- Remote debugging

As the name implied, this enables developers to validate code which has been deployed to a remote server. The remote option means files located on a remote server can be debugged using the Zend Studio Debugger and all information and the debug process will be controlled and supervised by the Zend Studio Client.

Debugging mode can be set by the preferences options: **Tools** → **Preferences** → **Debug**.

Debugging connectivity can be also tested by: **Tools** → **Check Debug Server Connection**. That option is shown below on Figure 3-12.

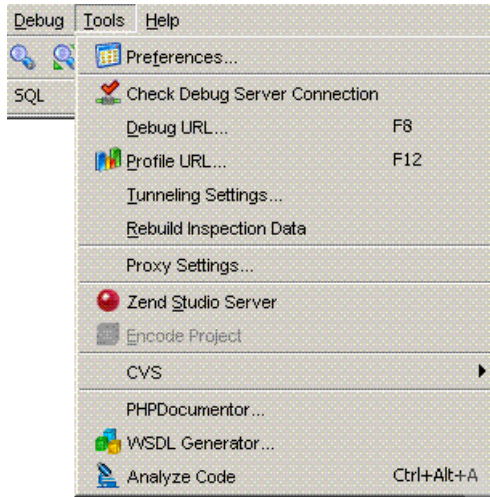


Figure 3-12 Zend Studio Client Tools option window

Zend Studio Client comes with a debugger step-by-step tutorial. To activate the tutorial, click: **Help** → **Tip of the day** → **Debug Demo**.

Here we shows a few debugging steps to get you started:

1. Create new project.
2. Copy the *Example 3-1* code into the editor window:

*Example 3-1 Example code used for debugging test*

---

```
<?
    $number_of_repeat = 5;
    $filename = "test.txt";

    if (!$handle = fopen($filename, 'a+')) {
        echo "Cannot open file ($filename)";
        exit;
    }

    for($i=0;$i<$number_of_repeat;$i++){
        $content = date("H:i:s");
        echo $content;
        fwrite($handle,$content . "\n");
    }

    fclose($handle);
?>
```

---

3. Save the file.
4. Place the cursor at the beginning of the file and press SHIFT+F10. This will start the debugger with the “Go to cursor” option. The first line of the script will be highlighted. This means that debugger is going to run that line of code. See Figure 3-13.

```
<?
$number_of_repeat = 5;
$filename = "test.txt";
```

Figure 3-13 Running the debugger: Line being investigated

5. In the debug window, the variables tab will fill out some default monitored values. You can continue the debugging process by pressing the F11 key (step into). Once the statement is executed, you check the variable values to see if they are what you expect.

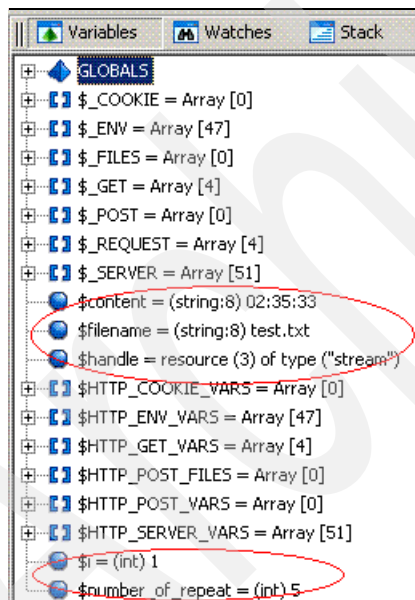


Figure 3-14 Variables tab

6. You also can set debugger to only watch certain variables under Watches window. Watches can be added dynamically during program execution by pressing SHIFT+F8. See Figure 3-15.

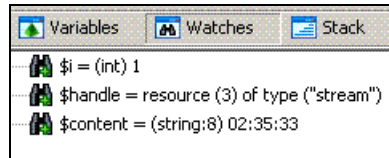


Figure 3-15 Watches debug window

7. All information generated by the script and printed to the Web browser window can be found in “Debug Output Window”. The View can be changed between normal text output and HTML output. See Figure 3-16. In our test script, information generated by the script is stored in the file and sent to the browser as well.

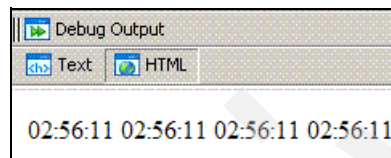


Figure 3-16 Debug Output window

8. Script execution can be always stopped by pressing SHIFT+F5.

For more information about Zend Studio, check the following:

Zend Studio Quick Start Guide:

[http://www.zend.com/content/download/1466/8580/version/1/file/Zend\\_Studio\\_Quick\\_Start\\_Guide.pdf](http://www.zend.com/content/download/1466/8580/version/1/file/Zend_Studio_Quick_Start_Guide.pdf)

Zend Studio User's Guide:

[http://www.zend.com/content/download/1586/9581/version/1/file/Zend\\_Studio\\_User\\_Guide.pdf](http://www.zend.com/content/download/1586/9581/version/1/file/Zend_Studio_User_Guide.pdf)





# PHP application development with DB2

This chapter discusses the installation, configuration, and development of applications which are supported by an Apache-powered Web server, PHP, and DB2 data server.

All of the components we describe in this chapter are no charge, from the Linux operating system to the DB2 Express-C database, which makes the deployment of Web applications on an enterprise level, scalable platform very easy.

This chapter describes the following:

- ▶ Installing and configuring the DB2 Client and Server
- ▶ Installing and configuring Apache, PHP, and DB2
- ▶ PHP and DB2 application development concepts

## 4.1 Application environment setup

In this section, we discuss the installation and configuration of the software necessary to get the application we developed up and running.

In our environment, we used two Linux servers. One machine hosted the database server, the second supported the Web server and application code along with the DB2 client for connecting to the database. This configuration is a typical setup, although the steps for configuring Apache, PHP, and DB2 are the same regardless of the server topology.

### 4.1.1 Lab environment description

Before we start the installation, we have to make sure that we have satisfied all the installation prerequisites for the software, including operating system version, kernel, packages, and compatible version of the Web server and database server and client.

We need to make decisions about the application environment, and for that we need to choose:

- ▶ Operating system
- ▶ Database management system
- ▶ Web server

We choose to use Linux, DB2 Express-C, and Apache HTTP server respectively.

With DB2, we have databases used by enterprises giving a robust and high performance database system. DB2 for Linux, UNIX, and Windows requires a minimum of 256 MB main memory. If you use graphical user tools, we recommend you have at least 512 MB main memory. If Linux is installed in the existing network, DB2 needs a static IP address.

#### Operating system considerations

For DB2 installation, the following Web site provides the hardware requirements and operating system prerequisites:

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

For Linux, validate your operating system configuration against the data provided in the following link:

<http://www.ibm.com/software/data/db2/linux/validate/>

In Linux, the `compat-libstdc++-33` package needs to be installed before installing DB2. If you intend to build Apache and PHP from the source code, you

will need the following tools: gcc, libgcc, cpp, flex, bison, libxml2, libxml2-devel, zlib-devel, readline-devel, make, tar, and gunzip. While installing the operating system, make sure that all these packages are installed. They are normally installed if you choose to install *Development Tools* or *Compilers and Tools*, or similar depending on the Linux distribution you are using.

We recommend for you to install and configure Apache and PHP yourself instead using the versions normally provided with the Linux distribution. You can uninstall these default packages using RPM. To find if the Apache HTTP Server is installed, use:

```
rpm -qa | grep httpd
```

To find if PHP packages are installed, use:

```
rpm -qa | grep php
```

To uninstall the package, use:

```
rpm -e <package name>
```

Once these steps are complete, we can install the software in the following order:

1. Install database server.
2. Install and configure database client.
3. Install and configure Apache and PHP.

## 4.1.2 User IDs and group

You need root privilege to perform all the installation tasks.

### **DB2**

DB2 installing wizard will create three users and their respective groups. They are:

- Instance owner

The DB2 instance owner holds the DB2 engine files and controls all DB2 processes spawned. The user also owns all file systems and devices used by the databases contained within the instance. The default instance in UNIX and Linux is *db2inst1* and the default group is *db2iadm1*.

- Fenced user

This user runs the stored procedure and user defined functions. DB2 supports stored procedure in languages, such as SQL, C, C++, Java, and .NET. For better security, these procedures and functions are run as a

different user. You can also run the stored procedure as the instance owner. The default user is *db2fenc1* and group is *db2iadm1*.

► DB2 Administration Server

This user hosts DB2 Administration Server, which is required for running administrative tools in Control Center and Task Center. There is only one Administration Server per machine since it can serve multiple instances in the same machine. The default user name is *dasusr1* and group is *dasadm1*.

### **Apache**

Apache by default runs as user *nobody* and group *nogroup* in Linux. In Windows, Apache is run as *SYSTEM* user. You can change this by changing the entry in `httpd.conf`. The entries are against the variable `User` and `Group` in the `httpd.conf` file. The following is the entry in the `httpd.conf` file:

```
User nobody
Group nobody
```

## **4.1.3 Database server installation and configuration**

Before starting the database installation in Linux, determine if you need to change kernel parameters. In Linux with Kernel Version 2.4.x, the default value for the message queue parameter should be changed to allow more simultaneous DB2 connections. The semaphore array parameters also need to be changed for DB2 to function properly. We can use `ipcs -l` to list present values of the parameters. Edit `/etc/sysctl.conf` and add the following lines to it:

```
kernel.msgmni = 1024
kernel.sem = 250 256000 32 1024
```

Issue `sysctl -p` after this to load these settings. Usually `sysctl.conf` gets loaded at startup by the network initialization script in some distributions. You may want to add it to system initialization files, such as `rc.local`.

You can install DB2 using either graphical DB2 Setup wizard or using command line. We recommend the graphical installation wizard. It is very simple and easy. Using the DB2 Setup wizard, DB2 takes care of all the user IDs and groups necessary for installation. You can choose the features to install. If you are connecting to a remote Linux machine, you must have X-window software capable of rendering a graphical user interface for the DB2 Setup wizard to run on your machine. Ensure that you have properly exported your display, for example, **`export DISPLAY=9.1.43.86:0`**.

If you have problems with your installation, you can find out why the installation failed in detail from the installation log files, `db2setup.his`, `db2setup.log`, and

db2setup.err, located, by default, in the /tmp directory. The db2setup.log file captures all DB2 installation information, including errors. The db2setup.his records all DB2 installations on your machine. DB2 appends the db2setup.log file to the db2setup.his file. The db2setup.err file captures any error output that is returned by Java Installer for DB2.

## Setting up the command line environment

Once DB2 is installed, you need to set up the DB2 command line environment by executing the db2profile script in <instance owners home directory>/sqllib/db2profile. You can add this to the .profile of the instance owner.

## DB2 network configurations

Our network uses TCP/IP protocol for communication. The DB2 client facilitates the connection between the application server and the database server which resides in another machine. DB2 uses industry standard Distributed Relational Database Architecture (DRDA) protocol over the network. DB2 supports almost all major network connection protocols. Do the following steps in the database server.

### 1. Updating the services file

Update the /etc/services file with an entry about service port name and the port number. In windows, this file is in %SystemRoot%\system32\drivers\etc.

**db2\_service 50000/tcp # Comment:DB2 listens through 50000 port**

### 2. Setting up the communication protocol

Update the communication protocol registry variable using db2set utility:

**db2set DB2COMM=tcpip**

### 3. Updating the database manager configuration parameters

The following DB2 command should be used to update the database manager configuration file. This parameter will make DB2 communicate using the respective port number or the respective port number entry in the services file if the db2\_service name is specified.

**db2 UPDATE DATABASE MANAGER CONFIGURATION USING SVCENAME  
<db2\_service/port\_number>**

You can check if the parameter is set using:

**db2 GET DATABASE CONFIGURATION | grep SVCENAME**

If you are configuring on the server edition of DB2, make sure that you restart DB2 by using the following commands.

**db2stop  
db2start**

## Verifying the installation

We can create a sample database using **db2samp1** and query the database using SQL to see whether the database is installed correctly or not and if everything is working fine.

```
db2samp1
db2 CONNECT TO SAMPLE
db2 SELECT * FROM STAFF
db2 CONNECT RESET
```

The **db2samp1** creates a database named **SAMPLE**. If the **SELECT** query returns a result set, we can consider the installation successfully completed.

## DB2 client installation and configurations

Installing DB2 Runtime Client is a fairly simple task, but make sure that you follow the steps while installing the DB2 server, including changing the kernel parameters. The client installation will create the default instance owner **db2inst1**.

### *Cataloging node in the client*

The information about the server machine needs to be cataloged in the client so that every time a database connection is made, the runtime client understands on which machine the database resides. For that, we use the following command:

```
db2 CATALOG TCPIP NODE <node_name> REMOTE <hostname/ip_address of the
server machine> SERVER <service_name/port_number>
```

**Note:** These are the minimum necessary parameters of this command. To get the full command syntax, you can use DB2 command line help.

```
db2 ? CATALOG TCPIP NODE
```

### *Cataloging the database at the node*

The information about on which node does the database reside is done using cataloging the database. Use the following command for that:

```
db2 CATALOG DATABASE <database name> AS <alias name> AT NODE <node name>
```

## Verifying client connection

After configuring the TCP/IP node, catalog the **SAMPLE** database in the client machine and try connecting to the **SAMPLE** database. By default, the authentication happens at the servers. While using the **CONNECT TO** command, make sure that you give the connection user name and password of the server.

```
db2 CONNECT TO sample USER <server_instancer> USING <server_password>
```

## Configuring an Apache Derby or IBM Cloudscape database

Cloudscape and DB2 Runtime Client are installed and configured automatically if you have installed Zend Core for IBM. You can also download and configure the Cloudscape environment separately.

We should have DB2 Runtime Client and Java installed to configure the Cloudscape database. You can download Cloudscape and Derby from the following Web sites:

[http://db.apache.org/derby/derby\\_downloads.html](http://db.apache.org/derby/derby_downloads.html)  
<http://www.ibm.com/software/data/Cloudscape/>

There is no charge to download, install, and use. Derby source code is available for free download under Apache Version 2 license.

For the DB2 Runtime Client, you can use the standalone DB2 Runtime Client, or Runtime Client included in DB2 Express-C or DB2 Data Server version.

Perform the following steps to configure Cloudscape/Derby Network Server.

1. Unzip the Derby package.
2. Set the environment variable `DERBY_INSTALL` to the path where the Derby package was unzipped.
3. Set the environment variable `JAVA_HOME` to the directory where Java is installed.
4. Go into `<DERBY_INSTALL>/frameworks/NetworkServer/bin` directory to set up the class path:

In Linux, if you have ksh installed, execute `setNetworkServerCP.ksh`; otherwise, set the class path mentioned in the `setNetworkServerCP.ksh` file manually.

In Windows, execute `setNetworkServerCP.bat`

5. Start the network server:

In Linux, if ksh is installed, run `startNetworkServer.ksh`

Otherwise, run:

```
java org.apache.derby.drda.NetworkServerControl start -h <host name>
-p <port number>
```

In Windows, use `startNetworkServer.bat`

6. Create the database.

Use `ij.ksh` in Linux or `ij.bat` in Windows to invoke the Derby command line called interactive JDBC (ij). Issue the following command to create a new database DLRSHF:

```
ij> connect  
'jdbc:derby:net://localhost:1527/dlrshp;create=true:user=db2inst1;password=123;';
```

7. Catalog the Derby network server with the DB2 Runtime Client using the DB2 command line processor (CLP):

```
db2 CATALOG TCPIP NODE derbyn timer REMOTE localhost SERVER 1527
```

8. Catalog the database with the DB2 Runtime Client using CLP:

```
db2 CATALOG DB dlrshp AT NODE derbyn AUTHENTICATION SERVICE
```

9. Verify connection to the database using CLP:

```
db2 connect to dlrshp user db2inst1 using 123
```

#### 4.1.4 Apache and PHP installation and configuration

There are several ways to install PHP with an Apache-powered Web server and support for DB2. This section describes a few of the most common ways for both Linux and Windows. Each of the methods has particular strengths so you should choose one based on flexibility or which features you need.

##### Installation in Linux

In Linux, there are three common approaches to installing the necessary software:

- Install Apache and Zend Core for IBM.
- Build PHP as a shared module for Apache.
- Build PHP statically with Apache.

##### *Install Apache and Zend Core for IBM*

The installation and configuration of Apache and Zend Core are discussed in Chapter 3, “Zend installation and configuration” on page 75.

**Note:** At the time of writing, the latest version of Zend Core for IBM 1.3.1 does not include PDO\_IBM/PDO\_ODBC. If you need use PDO in your application, you have to build PHP from source with the PDO\_IBM/PDO\_ODBC extension as described in the other two installation sections.

PDO\_IBM and PDO\_ODBC are included in Zend Core for IBM 2.0.1 and above.

##### *Build PHP as a shared module for Apache*

This is the most flexible way to build PHP with its extensions and integrate it with Apache. This method installs PHP as a module which can be loaded outside of



the main Apache process when Apache is configured to use it. Refer the Web site below for a more detailed explanation of the steps required to build Apache and PHP from source and how to enable other available extensions.

<http://www.php.net/manual/en/install.unix.php>

There are four extensions available to interface with DB2 and Cloudscape from PHP:

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

You can install all four extensions using the following steps:

1. Installing Apache 2:

- a. Download the source code from:

<http://httpd.apache.org/download.cgi>

- b. Then run the following commands:

```
tar -zxvf httpd-2.0.55.tar.gz
cd httpd-2.0.55
./configure --enable-module=so
make
make install
```

**Tip:** If you are using Apache 1.3, the source archive will be named `apache_1.3.34.tar.gz` or similar, instead of `httpd-2.0.55.tar.gz` for Apache 2. If you are using IBM HTTP Server (IHS) 1.3 or 6, use the GUI to install. For a comparison of which Web server to choose, see 1.3, “HTTP Servers” on page 11.

2. Installing and configuring PHP

- a. Download the source code from:

<http://www.php.net/downloads.php>

- b. Download the `ibm_db2` PECL extension from:

[http://pecl.php.net/package/ibm\\_db2](http://pecl.php.net/package/ibm_db2)

- c. Download the `PDO_IBM` PECL extension from:

[http://www.pecl.php.net/package/PDO\\_IBM](http://www.pecl.php.net/package/PDO_IBM)

- d. Extract the PHP source package.

```
tar -zxvf php-5.1.2.tar.gz
```

- e. Move the `ibm_db2` archive to the `php/ext` directory.
- ```
mv ibm_db2-1.2.0.tgz php-5.1.2/ext
```
- f. Extract the `ibm_db2` source package and rebuild the configuration script.
- ```
cd php-5.1.2/ext
gzip -d < ibm_db2-1.2.0.tgz | tar -xvf -
mv ibm_db2-1.2.0 ibm_db2
cd ..
rm configure
./buildconf --force
```
- g. Ensure that the `ibm_db2` extension is now available to be built.
- ```
./configure --help | grep ibm_db2
```
- It should return the following:
- ```
--with-IBM_DB2=DIR Include IBM DB2 Universal Database and Cloudscape support.
```
- h. Move the `PDO_IBM` archive to the `php/ext` directory.
- ```
mv PDO_IBM-1.1.0.tgz php-5.1.2/ext
```
- i. Extract the `PDO_IBM` source package and rebuild the configuration script.
- ```
cd php-5.1.2/ext
gzip -d < PDO_IBM-1.1.0.tgz | tar -xvf -
mv PDO_IBM-1.1.0 pdo_ibm
cd ..
rm configure
./buildconf --force
```
- j. Ensure that the `PDO_IBM` extension is now available to be built.
- ```
./configure --help | grep pdo_ibm
```
- It should return the following:
- ```
--with-pdo-ibm=DIR Include PDO IBM support, DIR is the base
```
- k. Now configure PHP.
- ```
./configure \
--with-IBM_DB2=/opt/IBM/db2/V8.1 \
--with-pdo-ibm=/opt/IBM/db2/V8.1 \
--with-pdo-odbc=ibm-db2,/home/db2inst1/sqllib \
--with-ibm-db2=/opt/IBM/db2/V8.1 \
--with-apxs2=/usr/local/apache2/bin/apxs
```
- Where:
- `--with-IBM_DB2` is for `ibm_db2` extension
  - `--with-pdo-ibm` is for `PDO_IBM` extension
  - `--with-pdo-odbc=ibm-db2` is for `PDO_ODBC` extension
  - `--with-ibm-db2` is for Unified ODBC extension

- Path `/opt/IBM/db2/V8.1` is the default path for DB2 libraries.

If you do not need any particular extension, you can remove that and install.

**Tip:** If you are using Apache 1.3, change the `apxs` flag to `--with-apxs=/usr/local/apache/bin/apxs`. If you are using IHS 1.3, change it to `--with-apxs=/opt/IBMHttpServer/bin/apxs`. If you are using IHS 6, change it to `--with-apxs2=/opt/IBMIHS/bin/apxs`.

You may want to add other configuration options at this point, although you can always rerun the configure and following make steps in the future. To see a list, type:

```
./configure --help
```

- I. Build and install PHP.

```
make
make install
```

Once the `make install` step completes, PHP is installed. Now we need to configure the PHP configuration file (`php.ini`) file to change the default setting for the extensions. We may also need to change the Apache configuration file (`httpd.conf`) so that Apache correctly loads the PHP module.

Confirm that the PHP installation has updated `httpd.conf`. The `LoadModule` path should point to the PHP shared module and `AddModule` says which is the module:

- Confirm that `httpd.conf` contains the following lines:

```
LoadModule php5_module modules/libphp5.so
AddModule mod_php5.c
```

- To enable files with a `.php` extension to parse, add the following line to `httpd.conf`:

```
AddType application/x-httpd-php .php
```

- To enable PHP files to be default files when only a directory is given in the URL, change the `DirectoryIndex` line to read:

```
DirectoryIndex index.html index.php
```

- Finally, you need to edit `apachectl` to inherit the DB2 environment. Add this line:

```
./ /home/db2inst1/sqllib/db2profile
```

**Tip:** The `apachectl` utility will be in the corresponding location depending on which Web server you are using:

- ▶ Apache 1.3: `/usr/local/apache/bin/apachectl`
- ▶ Apache 2: `/usr/local/apache2/bin/apachectl`
- ▶ IBM HTTP Server 1.3: `/opt/IBMHttpServer/bin/apachectl`
- ▶ IBM HTTP Server 6: `/opt/IBMIHS/bin/apachectl`

You should now restart Apache to pick up the configuration changes and verify the installation in 4.1.5, “Environment verification” on page 110.

### ***Build PHP statically with Apache***

This method creates an Apache executable with PHP built-in. This has some performance benefits, but PHP cannot be disabled as a module via the configuration file. Additionally, you will need to recompile Apache if you want to upgrade PHP.

Use the following steps to statically compile Apache 2 and PHP with DB2 extensions:

**Important:** This method can only be used with Apache source installations. IHS is only provided as a binary and cannot be recompiled.

#### 1. Configure Apache.

```
tar -zxvf httpd-2.0.55.tar.gz
cd httpd-2.0.55
./configure
```

**Note:** If you are using Apache 1.3, the source archive will be named `apache_1.3.34.tar.gz` or similar, instead of `httpd-2.0.55.tar.gz` for Apache 2. If you are using IHS 1.3 or 6, use the GUI to install.

#### 2. Configure and install PHP.

##### a. Extract the PHP source package.

```
tar -zxvf php-5.1.2.tar.gz
```

##### b. Move the `ibm_db2` archive to the `php/ext` directory.

```
mv ibm_db2-1.2.0.tgz php-5.1.2/ext
```

##### c. Extract the `ibm_db2` source package and rebuild the configuration script.

```
cd php-5.1.2/ext
gzip -d < ibm_db2-1.2.0.tgz | tar -xvf -
mv ibm_db2-1.2.0 ibm_db2
```

- ```

cd ..
rm configure
./buildconf --force

```
- d. Ensure that the `ibm_db2` extension is now available to be built.
- ```

./configure --help | grep ibm_db2

```
- It should return the following:
- ```

--with-IBM_DB2=DIR Include IBM DB2 Universal Database and Cloudscape support.

```
- e. Move the `PDO_IBM` archive to the `php/ext` directory.
- ```

mv PDO_IBM-1.1.0.tgz php-5.1.2/ext

```
- f. Extract the `PDO_IBM` source package and rebuild the configuration script.
- ```

cd php-5.1.2/ext
gzip -d < PDO_IBM-1.1.0.tgz | tar -xvf -
mv PDO_IBM-1.1.0 pdo_ibm
cd ..
rm configure
./buildconf --force

```
- g. Ensure that the `PDO_IBM` extension is now available to be built.
- ```

./configure --help | grep pdo_ibm

```
- It should return the following:
- ```

--with-pdo-ibm=DIR Include PDO IBM support, DIR is the bas

```
- h. Now configure and build PHP.
- ```

./configure \
--with-IBM_DB2=/opt/IBM/db2/V8.1 \
--with-pdo-ibm=/opt/IBM/db2/V8.1 \
--with-pdo-odbc=ibm-db2,/home/db2inst1/sqllib \
--with-ibm-db2=/opt/IBM/db2/V8.1 \
--with-apxs2=/usr/local/apache2/bin/apxs

```
- You may want to add other configuration options at this point, although you can always rerun the configure and following make steps in the future. To see a list, type:
- ```

./configure --help

```
- i. Build and install PHP.
- ```

make
make install

```
3. Configure Apache:
- Change directory back to Apache code to configure Apache.
- ```

cd apache_*

```

```
./configure --activate-module=src/modules/php5/libphp5.a  
make
```

The file libphp5.a does not exist, but it will be created after executing the make command. The created httpd binary file needs to be copied into the Apache bin directory.

4. Install the software.

```
make install
```

5. To enable files with a .php extension to parse, add the following line to httpd.conf:

```
AddType application/x-httpd-php .php
```

6. To enable PHP files to be default files when only a directory is given in the URL, change the DirectoryIndex line to read:

```
DirectoryIndex index.html index.php
```

7. Finally, you will need to edit apachectl to inherit the DB2 environment. Add this line:

```
. /home/db2inst1/sqllib/db2profile
```

**Note:** The apachectl utility will be in the corresponding location depending on which Web server you are using:

- ▶ Apache 1.3: /usr/local/apache/bin/apachectl
- ▶ Apache 2: /usr/local/apache2/bin/apachectl
- ▶ IBM HTTP Server 1.3: /opt/IBMHttpServer/bin/apachectl
- ▶ IBM HTTP Server 6: /opt/IBMIHS/bin/apachectl

You should now restart Apache to pick up the configuration changes and verify the installation in 4.1.5, “Environment verification” on page 110.

## Installation on Windows

Like the Linux environment, there are different ways to configure and use PHP, Apache, and DB2 in a Windows environment. You have different ways to set up the Windows environment:

- ▶ Install Apache and Zend Core for IBM.
- ▶ Install Apache, PHP, and appropriate DB2 interface DLLs.
- ▶ Compile Apache and PHP from source. Find the details here:  
<http://www.php.net/manual/en/install.windows.building.php>

### *Install Apache and Zend Core for IBM*

We discuss the installation and configuration of Apache and Zend Core in Chapter 3, “Zend installation and configuration” on page 75.

**Note:** At the time of writing, the latest version of Zend Core for IBM 1.3.1 does not include PDO\_IBM and PDO\_ODBC. If you need use PDO in your application, you will have to download the PDO DLLs listed in the section below. Substitute C:\Program Files\Zend\Core for IBM\lib\phpext for C:\PHP\ext and C:\Program Files\Zend\Core for IBM\etc\php.ini for C:\php\php.ini.

PDO\_IBM and PDO\_ODBC are included in Zend Core for IBM 2.0.1 and above.

### ***Install Apache, PHP, and DB2 interfaces***

You install and configure Apache and PHP from the Windows binaries. Refer the below Web site:

1. Installing Apache 2:
  - a. Download the Windows binary installer from:  
<http://httpd.apache.org/download.cgi>
  - b. Double-click apache\_2.0.55-win32-x86-no\_ssl.msi and follow the directions on the window.

**Tip:** If you are using IHS 6, use the GUI to install. For a comparison of which Web server to choose, see 1.3, “HTTP Servers” on page 11.

2. Installing and configuring PHP.
  - a. Download the Windows ZIP package (not the installer) from:  
<http://www.php.net/downloads.php>
  - b. Extract php-5.1.2-Win32.zip to C:\PHP.
  - c. Copy C:\php\php.ini-recommended to C:\php\php.ini..
  - d. Confirm that httpd.conf contains the following lines:  

```
LoadModule php5_module "c:/php/php5apache2.dll"  
AddModule mod_php5.c
```
  - e. To enable files with a .php extension to parse, add the following line to httpd.conf.  

```
AddType application/x-httpd-php .php
```
  - f. Add the location of php.ini.  

```
PHPIniDir "C:/php"
```

- g. To enable PHP files to be default files when only a directory is given in the URL, change the `DirectoryIndex` line to read:

```
DirectoryIndex index.html index.php
```

**Note:** The configuration steps above are the same for Apache 2 or IBM HTTP Server 6. The locations of the configuration file by default on Windows machines are:

Apache 2: C:\Program Files\Apache Group\Apache2\conf  
IBM HTTP Server 6: C:\Program Files\IBM HTTP Server\conf

► Installing the DB2 interfaces

The Unified ODBC extension comes with PHP by default. The `ibm_db2`, `PDO_IBM`, and `PDO_ODBC` libraries can be downloaded from the following URL. Make sure that you get the correct DLL for your PHP version.

<http://pecl4win.php.net/list.php>

- a. Download the following DLLs to C:\PHP\ext. If you are using PHP 5.1, the PDO DLLs will already exist: `php_ibm_db2.dll`, `php_pdo.dll`, `php_pdo_ibm.dll`, and `php_pdo_odbc.dll`
- b. Update C:\php\php.ini to load the extensions

```
extension_dir="c:\php\ext"  
extension=php_ibm_db2.dll  
extension=php_pdo.dll  
extension=php_pdo_ibm.dll  
extension=php_pdo_odbc.dll
```

You can now start Apache by running either C:\Program Files\Apache Group\Apache2\bin\Apache.exe or C:\Program Files\Apache Group\Apache2\bin\ApacheMonitor.exe.

## 4.1.5 Environment verification

The environment with PHP, DB2, and Apache can be verified by running a short PHP program and seeing if it is connecting to the database. Copy Example 4-1 into a file in the `htdocs` directory inside the Apache installation and name it `connect.php`. Try connecting with a database cataloged into your DB2 client. The password and user name should correspond to the database given.

*Example 4-1 PHP script to verify DB2, PHP, and Apache environment*

```
<?php  
$db_name = 'sample';  
$usr_name = 'db2inst1';  
$password = '123456';
```



```

$conn_resource = db2_connect($db_name, $usr_name, $password);

if ($conn_resource) {
    echo 'Connection to database succeeded.';
    db2_close($conn_resource);
} else {
    echo 'Connection to database failed.<br />';
    echo 'SQLSTATE: ' . db2_conn_error() . '<br />';
    echo 'Message: ' . db2_conn_errormsg(). '<br />';
}
?>

```

---

## Troubleshooting

If the script above does not return “Connection to database succeeded“, when you open <http://localhost/connect.php>, you can diagnose the problem based on the following possible issues.

**Tip:** By default, PHP reports error messages to the Apache error log file (logs/error\_log on Linux, logs/error.log on Windows). During development, it is easier to debug using error messages displayed in your Web browser. To toggle this option, change the following value in php.ini.

In production use: **display\_errors = Off**

In development use: **display\_errors = On**

Table 4-1 shows a list of possible errors.

*Table 4-1 Troubleshooting DB2, Apache, and PHP connection problems*

Result	Likely error and solution
Apache returns the script above as is in the browser or prompts you to download the file.	Apache not configured to process PHP files. Make sure you have added the AddType line to httpd.conf. AddType application/x-httpd-php .php
“Call to undefined function db2_connect()“ is showing in the browser or /usr/local/apache2/logs/error_log	The ibm_db2 extensions are not available to your PHP build. Make sure to compile PHP as shown in 4.1.4, “Apache and PHP installation and configuration” on page 102.

<p>Connection to database failed. SQLSTATE: Message:</p>	<p>The user who started Apache does not have the DB2 environment available. No specific error messages or SQLSTATEs are given because PHP cannot locate DB2. Add the following line to apachectl or root's .profile</p> <pre>. /home/db2inst1/sqllib/db2profile</pre> <p>An example error message from PDO is "PDOException with message 'SQLSTATE[] SQLSetEnvAttr: ODBC3: 0'".</p>
<p>Connection to database failed. SQLSTATE: 58031 Message: [IBM][CLI Driver] SQL1031N The database directory cannot be found on the indicated file system. SQLSTATE=58031 SQLCODE=-1031</p>	<p>Make sure you have created the SAMPLE database. If not, run: db2 =&gt; CREATE DATABASE SAMPLE</p>
<p>Connection to database failed. SQLSTATE: 08001 Message: [IBM][CLI Driver] SQL1032N No start database manager command was issued. SQLSTATE=57019 SQLCODE=-1032</p>	<p>The database is not running. Issue: db2 =&gt; db2start</p>
<p>Connection to database failed. SQLSTATE: 08001 Message: [IBM][CLI Driver] SQL1013N The database alias name or database name "xxxxx" could not be found. SQLSTATE=42705 SQLCODE=-1013</p>	<p>Ensure that the database name is correct.</p>

Connection to database failed. SQLSTATE: 08001 Message: [IBM][CLI Driver] SQL30082N Attempt to establish connection failed with security reason "24" ("USERNAME AND/OR PASSWORD INVALID"). SQLSTATE=08001 SQLCODE=-30082	You are not authenticating against the database. Ensure that your username and password are correct.
---	--

## 4.2 Using PHP with DB2 database

In this section, we describe the application programming interfaces in PHP for DB2. The available native interfaces for DB2 are:

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

### 4.2.1 `ibm_db2`

Before you start to use `ibm_db2`, make sure that `php.ini` has been set to the DB2 instance you want the PHP to use, so PHP will refer to the libraries of the respective instance for connecting and querying the database. This option is ignored in Windows. In Linux/UNIX, it overrides the environment variable `DB2INSTANCE`.

If not already present, you can make an entry in the `php.ini` file as follows:

```
[ibm_db2]
ibm_db2.instance_name=db2inst1
```

The default instance created in Linux/UNIX is `db2inst1`.

The DB2 command `db2ilist` lists the available instances in the machine. To find the current instance, use the DB2 command:

```
db2 get current instance
```

Another global variable that 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. It can have three values.

```
[ibm_db2 binary data handling]
; n can have 3 values as 1,2,3
ibm_db2.binmode = "n"
```

When set to “1”, then the DB2\_BINARY constant gets set and all binary data is handled as it is. When set to “2”, then the DB2\_CONVERT constant gets set and all the binary data is converted into ASCII by the ibm\_db2 driver. When set to “3”, then the DB2\_PASSTHRU constant gets set and all the binary data gets converted into a null value.

## Program flow

The program includes a special variable called a *resource* which refers to a handler responsible for connecting to the database, querying the database, and getting result sets from the query. In the program, we flow through two kinds of resources and they are:

- ▶ Connection resource
- ▶ Statement resource

Once we supply the connection credentials to the connecting function, if a connection is successful, the function returns a *connection resource* which can be used to execute multiple queries. After which, *statement resource* is returned, which can be used to retrieve the meaningful data from the database.

The steps taken by the PHP program during the transaction processing are:

1. Connect to the database.
2. Prepare and execute the statement.
3. Process the results.
4. Free the resources.

## Connect to the database

There are two methods of connecting with 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 connection resource is set to NULL, or the script ends. Performance can be impacted if database sessions are made and freed too often. But it is advisable to go for a non persistent connection when you are doing some INSERT, UPDATE, or DELETE 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. If PHP is configured as a CGI wrapper, we do not get the advantage since the instance of PHP interpreter itself is destroyed on every call of PHP page. Whenever we use persistent

connections to the database, it is good that we have transactions which are either a read-only application or with AUTOCOMMIT set to ON. Otherwise, it can result in many locks being made on the database resources because of uncommitted transactions, and, therefore, affect the application performance.

The functions `db2_pconnect` and `db2_connect` have a one-to-one mapping with each other in terms of parameters being passed. The functions can be used to connect to databases which are cataloged in the local DB2 client or to a remote machine by giving the fully qualified connection details.

### ***Catching connection errors***

The PHP application should be able show the user the correct message and code from the database server to explain why the connection was not made, or why an existing connection was broken. We have two functions for this:

- ▶ `db2_conn_error` returns the SQLSTATE value of the error
- ▶ `db2_conn_errormsg` returns the error message and the SQLCODE value from the database server.

The difference between SQLSTATE and SQLCODE is that the SQLSTATE values have almost the same meaning across different databases. SQLCODE differs between different database servers. In the case of SQLCODE, if you are using DB2 Connect™ to connect to a host system or System i™ or iSeries server, DB2 Connect maps the SQLCODE values to DB2 where it is cataloged. Once you get the SQLSTATE or SQLCODE, you can search the Information Center or execute the following command in the DB2 command line to get the reason why the connection failed and find out how the user should respond.

```
db2 "? SQLSTATE"
```

### ***Creating a connection to a cataloged database***

For the DB2 client, the database server for the cataloged database can be DB2 for Linux, UNIX, and Windows, DB2 for iSeries, or DB2 for z/OS. The cataloged database can be a Cloudscape/Derby database. Example 4-2 shows an example for connecting to the DLRSH database. For the cataloged database, the database alias name, user ID, and password of the database server are required parameters in `db2_connect`.

#### *Example 4-2 Connecting to a cataloged database*

---

```
<?php
$db_name = 'DLRSH';
$usr_name = 'db2inst1';
$password = '123456';

// For persistent connection, change db2_connect to db2_pconnect
$conn_resource = db2_connect($db_name, $usr_name, $password);
```

```

if ($conn_resource) {
    echo 'Connection to database succeeded.';
    db2_close($conn_resource);
} else {
    echo 'Connection to database failed.';
    echo 'SQLSTATE value: ' . db2_conn_error();
    echo 'with Message: ' . db2_conn_errormsg();
}
?>

```

---

### ***Creating a connection to 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 following format:

```

DRIVER={IBM DB2 ODBC DRIVER};DATABASE=database name;HOSTNAME=host
name;PORT=port;PROTOCOL=TCPIP;UID=user name;PWD=password;

```

Example 4-3 shows how to connect to DB2 using the non-cataloged method.

#### ***Example 4-3 Connecting to a non-cataloged database***

---

```

<?php
$db_name = 'DLRSHP';
$usr_name = 'db2inst1';
$password = '123456';
$hostname = 'radon.itsosj.sanjose.ibm.com';
$port = 50000;

$conn_string = "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=$db_name;" .
"HOSTNAME=$hostname;PORT=$port;PROTOCOL=TCPIP;UID=$usr_name;PWD=$password;";

// For persistent connection change db2_connect to db2_pconnect
$conn_resource = db2_connect($conn_string, '', '');

if ($conn_resource) {
    echo 'Connection to database succeeded.';
    db2_close($conn_resource);
} else {
    echo 'Connection to database failed.';
    echo 'SQLSTATE value: ' . db2_conn_error();
    echo 'with Message: ' . db2_conn_errormsg();
}
?>

```

---

## Handling transactions

DB2\_AUTOCOMMIT\_ON and DB2\_AUTOCOMMIT\_OFF are optional parameters in the connection function to change the autocommit status. Use associative PHP arrays with values DB2\_AUTOCOMMIT\_ON to set autocommit to on and DB2\_AUTOCOMMIT\_OFF to set autocommit to off. If nothing is set, the default is DB2\_AUTOCOMMIT\_ON. See Example 4-4.

### Example 4-4 Working with AUTOCOMMIT ON

---

```
<?php
$db_name  = 'DLRSH';
$usr_name = 'db2inst1';
$password = '123456';

// We do not need the below line. Default is auto commit
$options  = array('autocommit' => DB2_AUTOCOMMIT_OFF);

// For persistent connection change db2_connect to db2_pconnect
$conn_resource = db2_connect($db_name, $usr_name, $password, $options);

if ($conn_resource) {
    echo 'Connection to database succeeded.';
    if (db2_autocommit($conn_resource, DB2_AUTOCOMMIT_ON)) {
        echo 'Autocommit is set to on.';
        // Here we do the database manipulation
    } else {
        echo 'Autocommit set to off.';
    }
    db2_close($conn_resource);
} else {
    echo 'Connection to database failed.';
    echo 'SQLSTATE value: ' . db2_conn_error();
    echo 'with Message: ' . db2_conn_errormsg();
}
?>
```

---

You can use the `db2_autocommit` function to see the current value of autocommit and set the value of autocommit. When `db2_autocommit()` receives only the connection parameter, it returns the current state of AUTOCOMMIT for the requested connection as an integer value. A value of 0 indicates that AUTOCOMMIT is off, while a value of 1 indicates that AUTOCOMMIT is on. When `db2_autocommit()` receives both the connection parameter and autocommit parameter (DB2\_AUTOCOMMIT\_ON or DB2\_AUTOCOMMIT\_OFF), it attempts to set the AUTOCOMMIT state of the requested connection to the corresponding state; it returns TRUE on success or FALSE on failure.

If autocommit mode is set to off, you should commit the transaction using `db2_commit()` function. If the error occurs in the transaction, you can roll back that transaction using `db2_rollback()`.

Example 4-5 is a sample program with `db2_autocommit()`, `db2_commit()`, and `db2_rollback()`.

*Example 4-5 Transactions with auto commit set to OFF*

---

```
<?php
$db_name = 'DLRSHP';
$usr_name = 'db2inst1';
$password = '123456';
$options = array('autocommit' => DB2_AUTOCOMMIT_OFF);

// For persistent connection change db2_connect to db2_pconnect
$conn_resource = db2_connect($db_name, $usr_name, $password, $options);

if ($conn_resource) {
    echo 'Connection to database succeeded.';
    if (db2_autocommit($conn_resource)) {
        // This condition will not come since we have turned autocommit off
        echo 'Autocommit is on.';
    } else {
        echo 'Autocommit set to off.';
        // Here we do the database manipulation
        // db2_commit($conn_resource); // if no errors
        // db2_rollback($conn_resource); // if the errors happened
    }
    db2_close($conn_resource);
} else {
    echo 'Connection to database failed.';
    echo 'SQLSTATE value: ' . db2_conn_error();
    echo 'with Message: ' . db2_conn_errormsg();
}
?>
```

---

## **Prepare and execute the statement**

Before you start to prepare and execute the 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

### ***Type of cursor to be used***

PHP with `ibm_db2` supports two kinds of cursors.



- Forward only cursors

This is the default cursor of an PHP application with ibm\_db2. The cursor fetches the result set row by row in a unidirectional way. It is the ideal cursor when we do the read-only operations against the database.

- Scrollable cursors

The ibm\_db2 implements the scrollable cursor using keyset-driven scrollable cursor. This cursor can detect changes and make changes to the underlying data. When the cursor is opened, DB2 makes a keyset where it stores the keys, which is used to determine the order and 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.

### ***How to catch an error***

The application you write should be good enough that it catches and explains 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 a error occurs.

```
$stmt = db2_exec($conn_resource, $sql);
if (!$stmt) {
    echo 'SQLSTATE value: ' . db2_stmt_error();
    echo 'with Message: ' . db2_stmt_errormsg();
}
```

### ***Isolation level to use***

Whenever an SQL statement is executed, there can be other concurrent SQL statements accessing the same data. An *isolation level* determines how data is locked or isolated from other transactions that try to access the same data. DB2 has four isolation levels:

- Repeatable Read isolation level (RR)
- Read Stability isolation level (RS)
- Cursor Stability isolation level (CS)
- Uncommitted Read isolation level (UR)

In all isolation levels, DB2 ensures that any row that is changed by an application process during a unit of work is not changed by any other application processes, until the unit of work is completed. That is, the transaction is either committed or rolled back. The isolation level determines the duration of row locking, and they are:

- RR: All rows are locked until the end of the transaction.
- RS: Rows qualifying the predicate condition are locked until the end of the transaction.

- ▶ CS: Only those rows whose cursor is positioned are locked.
- ▶ UR: No rows are locked unless the data is changing.

The UR isolation level should only be used in case of read-only-query applications. With UR isolation level, the query can read uncommitted data. The concurrency decreases and data integrity increases when you move the isolation level from UR to RR. The RR isolation level ensures maximum data integrity.

### ***Recommended Isolation levels***

This is the general rule for choosing the isolation level:

- ▶ Read-write transactions: Use RS if high data stability is required; otherwise, use CS.
- ▶ Read-only transactions: Use RR or RS if high data stability is needed; otherwise, use UR.

The isolation level affects the performance of the application and the chances of deadlock vary in different isolation levels. By default, DB2 uses cursor stability isolation level. You can change the isolation level of the queries in the PHP application, and therefore control the concurrency in applications.

You can use two methods to change the isolation level in a PHP program:

- ▶ Appending WITH clause in SQL

You can use WITH UR|CS|RS|RR at the end of the SQL, so that particular SQL runs in the specified isolation level. See Example 4-6.

#### ***Example 4-6 Set isolation level in PHP with ibm\_db2***

---

```
// 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);
```

---

In the above example, the query is run in UR isolation level.

- ▶ 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 Example 4-7.

#### ***Example 4-7 Set isolation level in CURRENT ISOLATION special register***

---

```
// With connection being made and connection resource is in $conn_resource
$currentiso = 'SET CURRENT ISOLATION LEVEL TO RR';
$sql = 'SELECT c_id FROM customer';
```

---

```

$stmt      = db2_exec($conn_resource, $currentiso);
$stmt      = db2_exec($conn_resource, $sql);

// Execute other SQL statements
$currentiso = 'SET CURRENT ISOLATION LEVEL TO CS';
$stmt      = db2_exec($conn_resource, $currentiso);

```

---

## Prepare and execute

In DB2, the execution of SQL with respect to the APIs used to execute the SQL, consists of two phases:

- ▶ **Prepare:**

In the prepare state, the SQL is parsed for syntactic and semantic errors, and then an optimized access plan is made for the SQL statement.

- ▶ **Execute:**

In the execute state, the access plan made during the prepare phase is used to query the database or to do database manipulations. The advantage of an execute phase is that if we have to execute the same SQL with different values to the parameters once or more, we do not need to go through the prepare process again.

In PHP with `ibm_db2`, we can do execute and prepare using a single step or different steps.

### ***Prepare and execute together***

Doing 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 one time. 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 the associative array as the third parameter.

```

$stmt = db2_exec($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. Prepare the SQL statement
2. Bind the parameters
3. Execute the query

### **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 from using SQL.

Example of SQL statement with parameter markers

```
SELECT c_name FROM CUSTOMERS WHERE c_id = ? and c_phone = ?
```

The parameter markers (represented by “?”) are variables in the `WHERE` clause of an SQL statement, part of a `CALL` to a stored procedure, or part of `VALUES` in an `INSERT` statement. These values are unknown while the statement is prepared. The values can be supplied to the database engine to retrieve the results in two ways:

- ▶ Binding the parameter using `db2_bind_param`
- ▶ Passing the parameter as an array

The `db2_prepare` function returns a statement resource if it succeeds; otherwise, it will return a `FALSE` value which you can investigate using `db2_stmt_error`.

The following are advantages of using prepared statements:

- ▶ **Performance:**  
Prepared statements can be executed many times with different parameter markers, therefore, saving computing resources.
- ▶ **Security:**  
It is more secure since the database checks the bound values every time a new parameter is bound to make sure the data type is matching its respective column or parameter definition, therefore it is secure from common vulnerabilities such as SQL injection.

Example 4-8 shows how to prepare and then execute passing parameters as an array.

*Example 4-8 Prepare and then execute with parameters as an array*

---

```
<?php
$insert = 'INSERT INTO customer (c_name, c_email) VALUES (?, ?)';
$customer = array('kiran', '1@123.com');
```

```
// Prepare the statement
$stmt = db2_prepare($conn_resource, $insert);
if (!$stmt) {
    echo 'The prepare failed. ';
    echo 'SQLSTATE value: ' . db2_stmt_error();
    echo 'with Message: ' . db2_stmt_errormsg();
} else {
    // Execute the statement with a parameter array
    $result = db2_execute($stmt, $customer);
    if(!$result) {
        echo 'The execute failed. ';
        echo 'SQLSTATE value: ' . db2_stmt_error();
        echo 'with Message: ' . db2_stmt_errormsg();
    }
}
?>
```

---

## Bind the parameters

Use the `db2_bind_param` function of PHP 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 for all statements, except inserting the large objects and the `CALL` statement which is used to call and execute stored procedures. 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. Example 4-9 shows preparing, binding, and executing the statement. Take care of these things before you bind the variables are:

1. The PHP variable name needs to be provided in between the double quotes (") and without the dollar sign (\$) "variable".
2. Check for the position variable of the bound parameters. The indexing should start from 1.
3. For variables other than `INTEGER` and `VARCHAR`, we recommend you use the data type specifier `DB2_BINARY`, `DB2_CHAR`, `DB2_DOUBLE`, or `DB2_LONG`.

Example 4-9 is an example for the bind involving two parameter markers, one of which is an integer and the other is a character type data.

*Example 4-9 Prepare, bind, and then execute*

---

```
<?php
$database = 'd1rshp';
```

```

$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);
    $id   = 100;
    db2_bind_param($stmt, 1, "id", DB2_PARAM_IN);
    db2_bind_param($stmt, 2, "name", DB2_PARAM_IN, DB2_CHAR);
    $name  = 'Kiran';
    $result = db2_execute($stmt);

    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 (!$result) {
        db2_rollback($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();
}
?>

```

---

In another scenario, for the DECIMAL data type, you need the parameter DB2\_LONG in db2\_bind\_param. In the example 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);

```

### Execute the query

Once you prepare the query, bind the parameter in db2\_bin\_param or pass the parameter as an array to db2\_execute which executed the statement. The

statement resource obtained after the query prepare and binding is passed as the input to `db2_execute`. If the parameters to be bound are a variable array, then the array variable also needs to be provided as the second parameter.

► **Execute with parameter array**

You can see Example 4-8 on page 122 for how to bind the values for parameter markers in the SQL using `db2_execute` to pass parameters as an array.

► **Execute with parameter**

You can just execute the statement if the parameters are already bound using `db2_bind`. You can find a sample program for this method in Example 4-9.

Once the query is executed, we can use the statement resource to get the result set using the following functions:

- `db2_fetch_array`
- `db2_fetch_assoc`
- `db2_fetch_both`
- `db2_fetch_object`
- `db2_fetch_row`

## Working with XML

DB2 and Cloudscape/Derby database adhere to the SQL/XML standards, which help convert the relational data into XML using SQL. This means that we can generate XML data from relational data, which is very useful in creating reports. Everything happens in the database layer, making the application free from problems relating to XML generation.

Once you use `XMLSerialize` and return the XML as a CLOB variable, it can be manipulated as another XML document using PHP and XML functions. Example 4-10 shows a sample program for generating an “Order Details” XML document for a customer.

*Example 4-10 Using SQL/XML in DB2*

---

```
<?php
$database = 'd1rshp';
$user      = 'db2admin';
$password  = 'db2inst1';
$conn      = db2_connect($database, $user, $password);

if ($conn) {

    $stmt = db2_exec($conn, 'SET CURRENT SCHEMA db2inst1');
    $statement = '
SELECT XMLSERIALIZE(
```

```

CONTENT
XMLELEMENT(
    NAME "OrderDocument",
    XMLATTRIBUTES( c_id AS "CustomerNumber",
        o_id AS "OrderId"),
    XMLELEMENT( NAME "CustomerDetails",
        XMLELEMENT( NAME "CustomerName", c_name),
        XMLELEMENT( NAME "CustomerEmail", c_email),
        XMLELEMENT( NAME "CustomerPhone", c_phone),
        XMLELEMENT(
            NAME "CustomerAddress",
            XMLFOREST( c_street as "Street",
                c_city as "City",
                c_state AS "State",
                c_zip AS "ZipCode",
                c_phone AS "PhoneNumber")
        )
    ),
    XMLELEMENT(
        NAME "OrderDetails",
        XMLATTRIBUTES( d_name AS "DealerName"),
        XMLELEMENT( NAME "Dates",
            XMLELEMENT( NAME "OrderDate", o_indate),
            XMLELEMENT( NAME "DeliveryDate", o_outdate)),
        XMLFOREST(o_id AS "OrderNumber",
            o_price AS "OrderPrice"),
        XMLELEMENT( NAME "VehicleDetails",
            XMLATTRIBUTES( v_id AS "VehicleId",
                v_price AS "VehiclePrice"),
            XMLFOREST(v_type AS "VehicleType",
                v_model AS "VehicleModel",
                v_year as "YearReleased",
                v_color AS "VehicleColor",
                v_package AS "VehiclePackage",
                v_ac as "AC",
                v_powerwindows AS "PowerWindows",
                v_automatic AS "Automatic",
                v_doors AS "Doors")
            )
        )
    ) AS CLOB(1000)) AS report
FROM dealer, customer, orders, vehicle
WHERE
f_c_id = c_id AND
f_d_id = d_id AND
f_v_id = v_id AND
c_id = 101;

// Prepare and execute the XML SQL statement

```



```

$stmt = db2_exec($conn,$statement);
if (!$stmt) {
    echo 'Execution failed. ';
    echo 'SQLSTATE value: ' . db2_stmt_error();
    echo 'with Message: ' . db2_stmt_errormsg();
}

// Prepare XML output
header('Content-Type: text/xml');
$doc = new DomDocument;

while ($object = db2_fetch_object($stmt)) {

    // Load the xml file into DOMDocument
    $doc = DOMDocument::loadXML($object->REPORT);

    // Print the file
    print $doc->saveXML();
}

// Release resources
db2_free_stmt($stmt);
$doc = null;
db2_close($conn);
} else {
    echo 'Connection to database failed.';
    echo 'SQLSTATE value: ' . db2_conn_error();
    echo 'with Message: ' . db2_conn_errormsg();
}
?>

```

---

The SQL we used in Example 4-10 on page 125 returns an XML document, which is typecast into a CLOB data type and used within the program. You can see the SQL with XML publishing function used in Example 4-11.

---

*Example 4-11 SQL/XML for generating order report*

---

```

SELECT XMLSERIALIZE(
    CONTENT
    XMLELEMENT(
        NAME "OrderDocument",
        XMLATTRIBUTES( c_id AS "CustomerNumber",
            o_id AS "OrderId"),
        XMLELEMENT( NAME "CustomerDetails",
            XMLELEMENT( NAME "CustomerName", c_name),
            XMLELEMENT( NAME "CustomerEmail", c_email),
            XMLELEMENT( NAME "CustomerPhone", c_phone),
            XMLELEMENT(

```

```

        NAME "CustomerAddress",
        XMLFOREST( c_street as "Street",
                   c_city as "City",
                   c_state AS "State",
                   c_zip AS "ZipCode",
                   c_phone AS "PhoneNumber")
        )
    ),
    XMLELEMENT(
        NAME "OrderDetails",
        XMLATTRIBUTES( d_name AS "DealerName"),
        XMLELEMENT(NAME "Dates",
                   XMLELEMENT( NAME "OrderDate", o_indate),
                   XMLELEMENT( NAME "DeliveryDate", o_outdate)),
        XMLFOREST(o_id AS "OrderNumber",
                   o_price AS "OrderPrice"),
        XMLELEMENT(NAME "VehicleDetails",
                   XMLATTRIBUTES( v_id AS "VehicleId",
                                   v_price AS "VehiclePrice"),
                   XMLFOREST(v_type AS "VehicleType",
                               v_model AS "VehicleModel",
                               v_year as "YearReleased",
                               v_color AS "VehicleColor",
                               v_package AS "VehiclePackage",
                               v_ac as "AC",
                               v_powerwindows AS "PowerWindows",
                               v_automatic AS "Automatic",
                               v_doors AS "Doors")
                   )
        ) AS CLOB(1000)) AS report
FROM dealer, customer, orders, vehicle
WHERE
    f_c_id = c_id AND
    f_d_id = d_id AND
    f_v_id = v_id AND
    c_id = 101;

```

---

For more details about SQL/XML, refer to the DB2 Information Center at:

<http://publib.boulder.ibm.com/infocenter/db2luw/v8/>

## SQL queries with large objects

Large objects are normally used as parameter markers in INSERT statements.

### ***Binding***

There are two ways to bind these parameters:

### ► PHP variable with just path

The files to be inserted into the database are normally stored in the file system. If we use PHP functions to open and load the data into a variable, it increases memory overhead. For that, we can directly bind the file name to the column using the DB2\_PARAM\_FILE parameter along with the bind. The third parameter in db2\_bind\_param is DB2\_PARAM\_FILE and not DB\_PARAM\_IN. The variable passed contains the file path for the large object file. If we are using the Windows environment, two of the backward slashes (\) should be used in the file name, such as "C:\\1.jpg". Refer to Example 4-12 for using this method for inserting large objects.

*Example 4-12 Inserting BLOB with bind path variable*

---

```
<?php
$filename = '/home/kiran/1.jpg';
$h_id     = '1000';
$f_v_id   = '1000';
$h_name    = 'venice';
$h_type    = 'sample';
$insert    = 'INSERT INTO photo (h_id, f_v_id, h_name, h_type, h_image) VALUES
(?, ?, ?, ?, ?)';

$stmt = db2_prepare($conn_resource, $insert);

if (!$stmt) {
    echo 'The prepare failed. ';
    echo 'SQLSTATE value: ' . db2_stmt_error();
    echo 'with Message: ' . db2_stmt_errormsg();
} else {
    db2_bind_param($stmt, 1, "h_id", DB2_PARAM_IN);
    db2_bind_param($stmt, 2, "f_v_id", DB2_PARAM_IN);
    db2_bind_param($stmt, 3, "h_name", DB2_PARAM_IN);
    db2_bind_param($stmt, 4, "h_type", DB2_PARAM_IN);
    db2_bind_param($stmt, 5, "filename", DB2_PARAM_FILE);

    $result = db2_execute($stmt);

    if (!$result) {
        echo 'The execute failed. ';
        echo 'SQLSTATE value: ' . db2_stmt_error();
        echo 'with Message: ' . db2_stmt_errormsg();
    }
}
?>
```

---

### ► PHP variable with loaded data

The binary large objects can be used as a parameter marker when it is available in a PHP variable. For this, we use the binding option `DB2_BINARY` as the binding parameter. Example 4-13 shows how to insert a large object in a PHP variable.

*Example 4-13 Inserting BLOB by binding the entire variable*

---

```
<?php
$filename = 'C:\\\\1.jpg';
$fh       = fopen($filename, 'r');
$photo    = fread($fh, filesize($filename));
$h_id     = '1000';
$f_v_id   = '1000';
$h_name    = 'venice';
$h_type   = 'sample';

$insert = 'INSERT INTO photo (h_id, f_v_id, h_name, h_type, h_image) VALUES (?,
?, ?, ?, ?)';

$stmt = db2_prepare($conn_resource, $insert);
if (!$stmt) {
    echo 'The prepare failed. ';
    echo 'SQLSTATE value: ' . db2_stmt_error();
    echo 'with Message: ' . db2_stmt_errormsg();
} else {
    db2_bind_param($stmt, 1, "h_id", DB2_PARAM_IN);
    db2_bind_param($stmt, 2, "f_v_id", DB2_PARAM_IN);
    db2_bind_param($stmt, 3, "h_name", DB2_PARAM_IN);
    db2_bind_param($stmt, 4, "h_type", DB2_PARAM_IN);
    db2_bind_param($stmt, 5, "photo", DB2_PARAM_IN, DB2_BINARY);

    $result = db2_execute($stmt);

    if (!$result) {
        echo 'The execute failed. ';
        echo 'SQLSTATE value: ' . db2_stmt_error();
        echo 'with Message: ' . db2_stmt_errormsg();
    }
}
?>
```

---

### **Fetching large objects**

You can fetching large objects with SQL, just as you can any other parameter in DB2. The fetched data is loaded into the format, depending on which type of fetch function you use. Example 4-14 shows fetching a BLOB variable from the table and displaying it into the browser.

```
<?php
$database = 'd1rshp';
$user      = '';
$password  = '';
$conn      = db2_connect($database, $user, $password);
$currentschema = 'SET CURRENT SCHEMA db2inst1';

if ($conn) {

    db2_exec($conn, $currentschema);

    $statement = 'SELECT h_image, h_type FROM photo WHERE h_id = 100';

    // Prepare and execute the SQL statement
    $stmt = db2_exec($conn, $statement);

    while ($object = db2_fetch_object($stmt)) {
        header('Content-Type: ' . $object->H_TYPE);
        echo pack('H*', bin2hex($object->H_IMAGE));
    }

    if (!$result) {
        echo 'Execution failed. ';
        echo 'SQLSTATE value: ' . db2_stmt_error();
        echo 'with Message: ' . db2_stmt_errormsg();
    }

    // Release resources
    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();
}
?>
```

---

## Creating and calling stored procedures

Stored procedures are server-side programs which help reduce network traffic and improve security, centralized administration, and maintenance. In DB2, you can write stored procedures in SQL, C, C++, Java and .NET languages. These server-side programs can be called from the PHP application. The stored procedure can have:

- IN parameter: Values passed to the stored procedure.

- ▶ OUT parameter: Values coming out of the procedure.
- ▶ INOUT parameter: Values passed and coming out of the stored procedure.
- ▶ One or more result sets returned.

The stored procedure can be created using the SQL statement and can be done from a PHP program.

### ***Creating a stored procedure using PHP***

The CREATE PROCEDURE SQL statement can be executed inside an PHP program. Once the procedure is created, it is registered in the server. After the procedure is registered in the database server, the stored procedure can be called from any client with the required promontories.

Example 4-15 shows an PHP script which creates a stored procedure getdetails. This stored procedure has a IN parameter called custid, an OUT parameter called dlrid, and an INOUT parameter called vehid. This stored procedure returns two result sets.

#### ***Example 4-15 Creating a stored procedure using PHP***

---

```
<?php
$database = 'dlrshp';
$user      = 'db2inst1';
$password = '123';
$conn_resource = db2_connect($database, $user, $password);

$createproc =
'CREATE PROCEDURE getdetails(IN custid INTEGER, OUT dlrid INTEGER, INOUT vehid
INTEGER)
LANGUAGE SQL
BEGIN
DECLARE c1 CURSOR WITH RETURN FOR
SELECT c_id FROM customer WHERE c_id > custid;
DECLARE c2 CURSOR WITH RETURN FOR
SELECT v_model FROM vehicle WHERE v_id = vehid ;
SELECT max(d_id) INTO dlrid FROM dealer;
SELECT max(v_id) INTO vehid FROM vehicle;
OPEN c1;
OPEN c2;
END';

if ($conn_resource) {

    $result = db2_execute($stmt, $createproc);

    if (!$result) {
        echo 'The execute failed. ';
        echo 'SQLSTATE value: ' . db2_stmt_error();
    }
}
```

```

        echo 'with Message: ' . db2_stmt_errormsg();
    }

    db2_close($conn);

} else {
    echo 'Connection to database failed.';
    echo 'SQLSTATE value: ' . db2_conn_error();
    echo 'with Message: ' . db2_conn_errormsg();
}
?>

```

---

### ***Calling the stored procedure***

Example 4-16 shows how to call a stored procedure in PHP with `ibm_db2`. The functions used are:

- ▶ **db2\_bind\_param:** For binding the parameters. The parameters used to bind are `DB2_PARAM_IN`, `DB2_PARAM_OUT`, or `DB2_PARAM_INOUT` with respect to the type of parameter which is bound in the stored procedure.
- ▶ **db2\_next\_result:** If the stored procedure returns multiple result sets, the function `db2_next_result` is used to get the result set rows after the first one. The function `db2_next_result` returns the new statement handle which can be used to fetch the rows for the next result set.

#### ***Example 4-16 Executing and retrieving results from a stored procedure***

---

```

<?php
$database = 'dlrshp';
$user      = 'db2inst1';
$password = '123';
$conn_resource = db2_connect($database, $user, $password);

if ($conn_resource) {
    $c_id = 100;
    $dlrid = 1000;
    $vehid = 100;
    $sql= 'CALL getdetails(?, ?, ?)';

    $stmt = db2_prepare($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($stmt, 1, "c_id", DB2_PARAM_IN);
        db2_bind_param($stmt, 2, "dlrid", DB2_PARAM_OUT);
        db2_bind_param($stmt, 3, "vehid", DB2_PARAM_INOUT);
    }
}

```

```

    $amount    = 0;
    $c_id      = 100;
    $quantity  = 100;
    $result    = db2_execute($stmt);

    if (!$result) {
        echo 'The execute failed. ';
        echo 'SQLSTATE value: ' . db2_stmt_error();
        echo 'with Message: ' . db2_stmt_errormsg();
    }
}

echo 'The OUT variable is: ' . $dlrid;
echo 'The INOUT variable is: ' . $vehid;

echo 'This is the first result set';
while ($row = db2_fetch_array($stmt)) {
    echo $row[0];
}

echo 'This is the second second result set';
$result = db2_next_result($stmt);

if ($result) {
    while ($row = db2_fetch_array($result)) {
        echo $row[0];
    }
} else {
    echo 'Connection to database failed.';
    echo 'SQLSTATE value: ' . db2_conn_error();
    echo 'with Message: ' . db2_conn_errormsg();
}
?>

```

---

## 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. There are functions which can be used to retrieve the metadata about the columns, which is returned by the result set.

The function `db2_num_rows` returns the number of rows that was 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'. It is always good to



depend on the boolean return values of the `db2_fetch_` function to browse until the end of the result set.

The function `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 dynamically generated queries and stored procedures. Example 4-17 shows how it can be used.

---

*Example 4-17 Using metadata functions*

---

```
$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();
    }
}
```

---

There is a set of metadata functions with prefix `db2_field_` which 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. Example 4-18 is an example.

---

*Example 4-18 Use of returned column information functions*

---

```
<?php
$database = 'd1rshp';
$user      = '';
$password  = '';
$conn      = db2_connect($database,$user,$password);
```

```

if ($conn) {

    $statement = 'SELECT * FROM customer WHERE c_id > ? OR c_name NOT LIKE ?';

    // Prepare the statement
    $stmt = db2_prepare($conn, $statement);

    // Retrieve metadata
    print 'Field display size: ' . db2_field_display_size($stmt, 1);
    print 'Field precision: ' . db2_field_precision($stmt, 1);
    print 'Field field name: ' . db2_field_name($stmt, 1);
    print 'Field display scale: ' . db2_field_scale($stmt, 1);
    print 'Field display width: ' . db2_field_width($stmt, 1);

    db2_close($conn);

} else {
    echo 'Connection to database failed.';
    echo 'SQLSTATE value: ' . db2_conn_error();
    echo 'with Message: ' . db2_conn_errormsg();
}
?

```

---

These values are the output of the program in Example 4-18:

```

Field display size: 50
Field precision: 50
Field field name: C_NAME
Field display scale: 0
Field display width: 50

```

The output of the program Example 4-18 shows the details of the second column of the query (column is indexed from zero; therefore, the column with index number 1 is the second column).

## Fetching the result sets

The `ibm_db2` driver allows you to get the rows of the result sets into the indexed data structure and even allows you to scroll through the result set one by one and fetch using a different function. The data structure into which the data is put depends on which function is used to retrieve the results.

### *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 returns a false if there are zero rows returned. It takes the statement resource as input.

```

while ($row = db2_fetch_array($stmt)) {
    printf("%d %s %s", $row[0], $row[1], $row[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. See the following 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']);
}

```

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 statement option:

The option DB2\_ATTR\_CASE can have one of three different 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.

The following example shows how the column names are forced to lower case and how it is used.

```

$stmt = db2_prepare($conn_resource, $sql, array('DB2_ATTR_CASE' =>
DB2_CASE_LOWER));

```

While doing the `db2_fetch_assoc`, we can use lower case column name when we execute the above statement. We show the `db2_fetch_assoc` usage below:

```
while ($row = db2_fetch_assoc($stmt)) {
    printf("%d %s %s", $row['c_id'], $row['c_name'], $row['c_email']);
}
```

### ***Using db2\_fetch\_both***

This function returns an array which is indexed by both the column number and the column name.

```
while ($row = db2_fetch_both($stmt)) {
    printf("%d %s %s", $row[0], $row['C_NAME'], $row['C_EMAIL']);
}
```

### ***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. The example below shows how it is used.

```
while ($row = db2_fetch_object($stmt)) {
    printf("%d %s %s", $row->C_ID, $row->C_NAME, $row->C_EMAIL);
}
```

### ***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. The following example 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)
    );
}
```

Example 4-19 shows how to use scrollable cursor and go to the row number specified in the `db2_fetch_row`. In DB2, the `ROWNUMBER() OVER(ORDER BY <column name>)` expression is used to get the row number of the row in the specified order. The example shows how the SQL is used.

#### ***Example 4-19 Using scrollable cursor***

---

```
$sql = 'SELECT (ROWNUMBER() OVER(ORDER BY c_id)) AS rownumber, c_id, c_name
FROM customer WHERE c_id >= ?';
$stmt = db2_prepare($conn_resource, $sql, array('cursor' => DB2_SCROLLABLE));
```

```

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();
    }
}

$row = db2_fetch_row($stmt, 1);
printf(
    "%d %d %s",
    db2_result($stmt, 0),
    db2_result($stmt, 1),
    db2_result($stmt, 2)
);

```

---

## Freeing the resources

You can free the statement resources using `db2_free_stmt` and the result set resources by 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 allocate persistent connections, the **db2\_close** request is ignored and the connection resource is used by the next similar connection.

```

$result = db2_close($conn_resource);
if ($result) {
    echo 'Connection was successfully closed.';
}

```

## 4.2.2 PDO\_IBM/PDO\_ODBC

In this section, we describe the use of PDO in conjunction with DB2 and Cloudscape databases. PHP Data Objects over ODBC (PDO\_IBM/PDO\_ODBC) provide another native database access layer for accessing the DB2 databases on Linux, UNIX, Windows, z/OS, and iSeries machines, and Cloudscape or

Apache Derby databases. PDO is based on open standards and the methods of accessing different database managers are the same. PDO, when compiled using db2 libraries, uses DB2 CLI (call level interface) to interact with DB2.

After the installation and configuration of DB2 and the PDO\_IBM or PDO\_ODBC extension along with PHP and Apache, the variables that can be edited in the `php.ini` file to make PDO work with DB2 are:

► **pdo\_odbc.db2\_instance\_name**

This entry has to be made in UNIX and Linux installations in `php.ini` in order to guide PHP to the instance where the DB2 libraries are located.

For PDO\_IBM:

```
[PDO_IBM instance name]
pdo_ibm.db2_instance_name= db2inst1
```

For PDO\_ODBC:

```
[PDO_ODBC instance name]
pdo_odbc.db2_instance_name= db2inst1
```

► **pdo\_odbc.connection\_pooling**

*Connection pooling* helps database application performance due to the reduction of new connections that are made as old connections are reused. It can be 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 connection pooling. We have three values that can be given to this variable based on which connection pooling is carried out.

– **Strict**

A connection is reused when the connection credentials exactly match the connection options of an existing closed connection. We recommend this option when connection pooling is enabled.

– **Relaxed**

Connections with a matching connection string are reused and not all connection attributes are checked, so that the connection can be reused.

– **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]
;can have values strict, relaxed, off
pdo_ibm.connection_pooling = relaxed
```

or

```
[PDO connection pooling]
;can have values strict, relaxed, off
pdo_odbc.connection_pooling = relaxe
```

► **pdo.dsn.\***

Data Source Name (DSN) contains the information for connectivity to a database through an ODBC or CLI driver. It contains 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 both cataloged and non-cataloged DB2. The following example shows the entry for the DSN alias:

For PDO\_IBM:

```
[DSN alias for pdo_db2]
pdo.dsn.dealerbase="ibm:dlrshp"
```

For PDO\_ODBC:

```
[DSN alias for pdo_db2]
pdo.dsn.dealerbase="odbc:dlrshp"
```

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");
```

## Program flow

PDO programming uses Object-Oriented concepts. Once the connections are made to the DB2 database, an instance of PDO base class is created which acts as a database handle, which is used when further activities are carried on against the database. Different objects that are created during the database activity can be classified as:

- Connection object (instance of PDO)
- Statement object (instance of PDOStatement)
- Exception (instance of PDOException)

The steps taken by a PDO program during transaction processing are:

1. Connect to the database.
2. Prepare and execute the statement.
3. Process the results.
4. Free the resources.

## Connect to the database

The connection to a database is made when the instance of the PDO object is made. The PDO object is made by calling the constructor of the class `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 databases, which includes the type of driver used (CLI/ODBC in the case of DB2), the name of data sources, and the connection credentials, such as user name and password. The DSN parameters required are depend on the connection made to the database (cataloged or non-cataloged).

If you are connecting to a DB2 or Cloudscape database, it is always better to go for cataloged connections.

### DSN for cataloged connection

For PDO\_IBM:

```
ibm:DSN=d1rshp;UID=db2inst1;PWD=123
```

For PDO\_ODBC:

```
odbc:DSN=d1rshp;UID=db2inst1;PWD=123
```

### DSN for non-cataloged connection

For PDO\_IBM:

```
ibm:DSN={IBM DB2 ODBC  
DRIVER};HOSTNAME=localhost;PORT=50000;DATABASE=d1rshp;PROTOCOL=TCPIP;UID=db  
2inst1;PWD=123;
```

For PDO\_ODBC:

```
odbc:DSN={IBM DB2 ODBC  
DRIVER};HOSTNAME=localhost;PORT=50000;DATABASE=d1rshp;PROTOCOL=TCPIP;UID=db  
2inst1;PWD=123;
```

An example of connecting using the cataloged method is given below:

```
<?php  
try {  
    /* Use one of the following connection string */  
    /* For PDO_IBM */
```



```

        $constrng = 'ibm:DSN=d1rshp;UID=db2inst1;PWD=123';
        /* for PDO_ODBC */
        $constrng = 'odbc:DSN=d1rshp;UID=db2inst1;PWD=123';
        $dbh = new PDO($constrng);
        echo 'Connected';
    } catch (PDOException $exp) {
        echo 'Exception: ' . $exp->getMessage();
    }
}
?>

```

The program to connect to the database, using non-cataloged connection, is:

```

<?php
try {
    /* Use one of the following connection string */
    /* For PDO_IBM */
    $constrng = 'ibm:DSN={IBM DB2 ODBC
    /* For PDO_ODBC */
    $constrng = 'odbc:DSN={IBM DB2 ODBC
    DRIVER};HOSTNAME=localhost;PORT=50000;DATABASE=d1rshp;PROTOCOL=TCPIP;UID
    =db2inst1;PWD=123';
    $dbh = new PDO($constrng);
    echo 'Connected';
} catch (PDOException $exp) {
    echo 'Exception: ' . $exp->getMessage();
}
?>

```

The port to which the DB2 server listens is mentioned in *port*. In Linux, its value can be found out or set using:

```
db2 get database manager cfg | grep SVCE
```

If it is a service name, you can look into `/etc/services` for the corresponding entry and find the port number.

In Windows, you can use **Start** → **Control Panel** → **Administrative Tools** → **Data Sources (ODBC)** to configure a DSN to point to the appropriate DB2 instance.

The DSN name can be given to a PDO program in three ways:

- As a parameter to the constructor

In this method, we have the DSN as a string in the program, but this makes the program dependent on the database. Every time the database base location or password is changed, the corresponding entry has to be changed in all programs that are referring to this database.

```

<?php
/* Use one of the following connection string */

```

```

/* The connection below is for pdo_ibm */
$constring = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constring = 'odbc:dlrshp';
try {
    $dbh = new PDO($constring, '', '');
    echo 'Connected';
} catch (PDOException $exp) {
    echo 'Exception: ' . $exp->getMessage();
}
?>

```

► In a file

In this method, the DSN is put into 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.

Create a file name `dsnfile` and store it in `/usr/local` directory. Use in:

```

<?php
$constring = 'uri:file:///usr/local/dsnfile';
try {
    $dbh = new PDO($constring, '', '');
    echo 'Connected';
} catch (PDOException $exp) {
    echo 'Exception: ' . $exp->getMessage();
}
?>

```

► Use aliasing in `php.ini` file

Refer to “`pdo.dsn.*`” on page 141.

As in `ibm_db2`, you can go for both persistent and non-persistent connections with `PDO_IBM/PDO_ODBC`. If the 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 if you 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.

```

<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constring = 'odbc:dlrshp';
try {

```

```

        $dbh = new PDO($constrng, 'db2inst1', '123');
        echo 'Connected';
    } catch (PDOException $exp) {
        echo 'Exception: ' . $exp->getMessage();
    }
?>

```

To make a persistent connection:

```

<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constrng = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constrng = 'odbc:dlrshp';
try {
    $dbh = new PDO($constrng, 'db2inst1', '123', array(PDO::ATTR_PERSISTENT
=> true));
    echo 'Connected';
} catch (PDOException $exp) {
    echo 'Exception: ' . $exp->getMessage();
}
?>

```

## Catching errors in PDO

Error handling in the PDO program could be better called exception handling. As in the case of any error, a signal, which is the exception, is thrown. We can have methods inside to catch the exception and display the details in order to find out what, and where something went wrong. The block of code which does this is called *exception handler*. We normally have a “try and catch” construct in PDO programs to handle exceptions in the best possible way.

When an exception is raised by your PDO program, an instance of `PDOException` is created. The `PDOException` class is the extension of `Exception` class and has three modes of operation. The error handling modes are forced by setting the PDO attribute `PDO::ATTR_ERRMODE`. The three values to the attribute are:

- ▶ **PDO::ERRMODE\_SILENT**

This is the default mode of error handling in PDO. The respective error code and messages are set in both the statement object (`PDOStatement`) and database object (`PDO`) when a error happens.

- ▶ **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 `E_WARNING`. You can suppress this by using a “@” in front of the function name.

## ► PDO::ERRMODE\_EXCEPTION

While setting this attribute, the PDO will throw an exception with maximum information and a stack trace.

You can see an example of setting the error handling mode below:

```
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constring = 'odbc:dlrshp';
$dbh = new PDO($constring, 'db2inst1', '123');
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

The methods associated with the PDO instance 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.

The best way to handle the connection specific error is by the use of the `try catch` statement. When an error happens, the `PDOException` is thrown and the details of the exception object is printed using the `getMessage()` method. See the example below:

```
<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constring = 'odbc:dlrshp';
try {
    $dbh = new PDO($constring, 'db2inst1', ' ');
} catch (PDOException $exp) {
    echo 'Exception: ' . $exp->getMessage();
}
?>
```

In the example, there was no password supplied so there should be an exception why the connection could not be made. The output looks like:

```
Exception : SQLSTATE[08001] SQLConnect: -30082 [IBM] [CLI Driver]
SQL30082N Attempt to establish connection failed with security reason
"3" ("PASSWORD MISSING"). SQLSTATE=08001
```

## Handling transactions

PDO, by default, runs in autocommit mode. That is, all queries are either committed or rolled back depending on it having passed or failed. You can use methods `PDO::beginTransaction` to make multiple SQL queries to be part of a 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`. If you have a huge transaction, and these transactions are doing insert, update, or delete operations, the system can have a large number of locks held up on the database objects that are changed. The possibility of deadlocks happening increases when the concurrent database accessing increases.

Example 4-20 shows a 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.

*Example 4-20 PHP with IBM\_PDO/PDO\_ODBC*

---

```
<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constring = 'odbc:dlrshp';
$dbh = new PDO($constring, 'db2inst1', '123');
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

try {
    $dbh->beginTransaction();
    $dbh->exec("
        INSERT INTO customer (c_name, c_email) VALUES ('Jen', 'a123@123.com')
    ");
    $dbh->exec("
        INSERT INTO customer (c_name, c_email) VALUES ('Kir', '1a23@123.com')
    ");
    $dbh->exec("
        INSERT INTO customer (c_name, c_email) VALUES ('Dan', '1n23@123.com')
    ");
    $dbh->commit();
} catch (PDOException $exp) {
    print 'Rolling back transaction';
    $dbh->rollback();
    print_r($dbh->errorInfo());
    echo 'Exception: ' . $exp->getMessage();
}
```

?>

---

## Prepare and execute the statement

Before we plan to execute the statement we have the following design considerations:

- ▶ Type of cursor used
- ▶ How to catch the error
- ▶ Isolation level to use

## Type of cursor to be used

Like ibm\_db2 extension, PDO also supports two kinds of cursors:

### ▶ Forward only cursor

This cursor is the default cursor in 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 could be forced by setting `PDO::ATTR_CURSOR` to `PDO::CURSOR_FWDONLY` in the `PDOStatement::prepare`.

```
$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));
    $sth->execute(array(':name' => 'Kiran'));
    print_r($sth->fetchAll());
} catch (PDOException $exp) {
    echo 'Exception: ' . $exp->getMessage();
}
```

### ▶ Scrollable cursor

We can specify the scrollable cursor by setting the `PDO::ATTR_CURSOR` to `PDO::CURSOR_SCROLL`. See example below.

```
$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();
}
```

```
}
```

## How to catch an error

As discussed in “Catching errors in PDO” on page 145, there are three error handling modes. Now you will see the differences among the three modes and the differences catching errors in the statement level.

### ► PDO::ERRMODE\_SILENT

In this mode, the error is not caught and the next statement executed after the function fails.

### ► PDO::ERRMODE\_WARNING

This raises an E\_WARNING. See Example 4-21. You can suppress this by using a “@” in front of the function name.

*Example 4-21 Exception handling using PDO::ERRMODE\_WARNING*

---

```
<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constring = 'odbc:dlrshp';
$dbh = new PDO($constring, 'db2inst1', '123');
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);

$sql = "SELECT c_id FROM customer WHERE notexistingcol = :name";
$stmt = $dbh->prepare($sql);
try {
    $sth->execute(array(':name' => 'Daniel'));
    $row = $sth->fetchAll();
    print_r($row);
} catch (PDOException $exp) {
    print_r($sth->errorInfo());
    echo 'Exception: ' . $exp->getMessage();
}
?>
```

---

The example will fail because the column referenced in the SQL predicate does not exist. The execute fails and the fetch also will fail since the program is trying to fetch the unprepared SQL. The error output for PDO\_ODBC looks like:

```
PHP Warning: PDOStatement::execute(): 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) in
/usr/local/apache2/htdocs/pdo/pdo_odbc/errstmt.php on line 9
PHP Warning: PDOStatement::fetchAll(): SQLSTATE[HY010]: Function
sequence error: -99999 [IBM][CLI Driver] CLI0125E Function sequence
error. SQLSTATE=HY010 (SQLFetchScroll[-99999] at
/root/Desktop/php-5.1.2/ext/pdo_odbc/odbc_stmt.c:371) in
/usr/local/apache2/htdocs/pdo/pdo_odbc/errstmt.php on line 10
```

#### ► 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 more function in the try block. The control goes to the catch block where the exception is handled. See Example 4-22. If you are using transactions with PDO::beginTransaction, if the statement fails, the rollback can be handled in the catch section.

*Example 4-22 Exception handling using PDO::ERRMODE\_EXCEPTION*

---

```
<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constring = 'odbc:dlrshp';
$dbh = new PDO($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 {
    $sth->execute(array(':name' => 'Whei-Jen'));
    $row = $sth->fetchAll();
    print_r($row);
    $sth->closeCursor();
} catch (PDOException $exp) {
    echo 'Exception: ' . $exp->getMessage();
}
?>
```

---

You can see that an exception is generated. The output for PDO\_ODBC looks like:

```
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)
```



### ***Isolation level to use***

Considerations of which isolation level to use in PHP with PDO\_IBM/PDO\_ODBC are the same as for the ibm\_db2 extension. You can refer to “Isolation level to use” on page 119. You can have the isolation level changed in the statement level using the WITH clause or you can change the CURRENT ISOLATION special register.

The execution of an SQL statement in PDO is carried out in two steps, preparation and execution. Refer to the “Isolation level to use” on page 119 for more details.

### **Executing a statement**

We have two choices for executing a statement, they are:

- ▶ Prepare and execute together
- ▶ Prepare and execute separately

#### ***Prepare and execute together***

PDO provides mechanisms by which you can prepare and execute in a single step. We have two kinds of SQL statements:

##### **▶ SQL statements returning result sets**

Normally, we use this method for execution of SQL statements which are not executed repeatedly in the same program. 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.

Example 4-23 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.

*Example 4-23 Prepare and execute SQL returning result set*

---

```
<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constring = 'odbc:dlrshp';
$dbh = new PDO($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();
    }
    ?>

```

---

### ► 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 for this type of SQL statement. The `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. See Example 4-24.

*Example 4-24 Prepare and execute SQL not returning result set*

```

<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constring = 'odbc:dlrshp';
$dbh = new PDO($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;
?>

```

---

### ***Prepare and then execute***

If you are executing the same SQL multiple times with different parameters, it is always better to use this method. It 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.

This way of executing SQL has three steps:

1. Preparing the SQL statement
2. Binding the parameters
3. Executing the statement

### **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 even set this option later using `setAttribute` method.

You can see how you set the cursor to scrollable.

```
$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` into `PDO::CURSOR_FWDONLY`.

The 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.

## Binding the parameters

There are two ways that you can bind the SQL statement: using a bind function or passing the values as the parameter in an array to the execute 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 types 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

We cannot use both named and nameless parameter markers in the same statement. Example 4-25 shows how to use `bindParam` and `bindValue` along with named parameter markers.

### *Example 4-25 Using bind function*

---

```
<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constring = 'odbc:dlrshp';
$dbh = new PDO($constring, '', '');
$cid = 100;
```

```

$name = 'Holger';
$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();
}
?>

```

---

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

Example 4-26 is an example for using the nameless parameter marker in a PDO program.

*Example 4-26* Binding by passing parameter values in an array

---

```

<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constring = 'odbc:dlrshp';
$dbh = new PDO($constring, '', '');
$cid = 100;
$name = 'Holger';
$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();
    }
    ?>

```

---

## 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.

Example 4-27 is an example for binding nameless parameter markers along with `PDOStatement::execute`.

### *Example 4-27 Using `PDOStatement::execute` with nameless parameter marker*

```

$sql = "select c_name from customer where c_id > ?";
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

try {
    $sth = $dbh->prepare($sql);
    $rc = $sth->execute(array(1, $id ,PDO::PARAM_INT));
    $result = $sth->fetchAll();
    print_r($result);
} catch (PDOException $exp) {
    print_r($sth->errorInfo());
    echo 'Exception: ' . $exp->getMessage();
}

```

---

Example 4-28 shows how to use a named parameter marker in the execute function.

### *Example 4-28 Using `PDOStatement::execute` with a named parameter marker*

```

<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constring = 'odbc:dlrshp';
$dbh = new PDO($constrng, '', '');
$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();
    }
?>

```

---

## 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.

Example 4-29 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.

*Example 4-29 Calling stored procedure in PHP with PDO\_IBM/PDO\_ODBC*

---

```

<?php
/* Use one of the following connection string */
/* The connection below is for pdo_ibm */
$constring = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constring = 'odbc:dlrshp';
$dbh = new PDO($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
$stmt->bindParam(2, $amount, PDO::PARAM_INT|PDO::PARAM_INPUT_OUTPUT);

// Input output parameter
$stmt->bindParam(3, $quant, PDO::PARAM_INT|PDO::PARAM_INPUT_OUTPUT);

$stmt->execute();

echo 'The INOUT parameter is: ' . $quant . ' and the OUT parameter is: ' .
$amount;
$rowset = $stmt->fetchAll(PDO::FETCH_NUM);
print_r($rowset);
} catch (PDOException $exp) {
    print_r($stmt->errorInfo());
    echo 'Exception: ' . $exp->getMessage();
}
?>

```

---

## Process the results

PDO offers two ways to fetch data. They are:

- ▶ **Buffered fetch**

This type of fetch allows you to fetch all the rows in the result set returned by an SQL query into an array. But if the query is to return a huge result set, we do not advise this as it will consume a lot of system resources.

- ▶ **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.

Example 4-30 shows an example for the fetch of bound columns.

#### ***Example 4-30 Fetching data using `FETCH_BOUND`***

---

```

<?php
/* Use one of the following connection string */

```

```

/* The connection below is for pdo_ibm */
$constring = 'ibm:dlrshp';
/* The connection below is for pdo_odbc */
$constring = 'odbc:dlrshp';
$dbh = new PDO($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();
}
?>

```

---

## Using Fetch

This is the default fetch option with no parameters passed to the fetch function. This function returns each row of the result set with the columns indexed by its position.

```

while ($row = $sth->fetch()) {
    echo 'id is: ' . $row[0] . ' and name is: ' . $row[1];
}

```

## 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.

```

while ($row = $sth->fetch(PDO::FETCH_ASSOC)) {
    echo 'id is: ' . $row['C_ID'] . ' and name is: ' . $row['C_NAME'];
}

```

## 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.

```

while ( $row = $sth->fetch(PDO::FETCH_BOTH)) {
    echo 'id is: ' . $row[0] . ' and name is: ' . $row['C_NAME'];
}

```



### **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. See Example 4-31.

*Example 4-31 Using scrollable cursors*

---

```
$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();
}
```

---

### **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.

```
while ($row = $sth->fetch(PDO::FETCH_LAZY)) {
    print_r($row->C_NAME);
}

while ($row = $sth->fetch(PDO::FETCH_OBJ)) {
    print_r($row->C_NAME);
}
```

### **Note**

When an SQL statement returns two or more columns with the same name from different tables, there is an ambiguity in fetching these result sets into an associative array using `db2_fetch_assoc` and `db2_fetch_both`. Both methods refer to the returned data using column names. Using `db2_fetch_object` to fetch the result sets to an object whose properties are column names also has the same problem.

In Example 4-32, both Customer and Dealer table have ID column. The output in the print statement may not be what you expect.

*Example 4-32 Select columns with the same column name*

---

```
$statement = "select c.id, d.id, c_email from customer c, dealer d";
$stmt      = db2_prepare($conn, $statement);
$result    = db2_execute($stmt);

while ($row = db2_fetch_assoc($stmt)) {
    echo 'id is: ' . $row['ID'] . ' ' . $row['ID'] . ' email is: ' .
$row['C_EMAIL'];
}

if (!$result) {
    echo 'The execute failed. ';
    echo 'SQLSTATE value: ' . db2_stmt_error();
    echo 'with Message: ' . db2_stmt_errormsg();
}
```

---

There are two workarounds for this problem:

- Use the correlation name

Change the SQL statement to give the columns with the same name a different correlation name. See Example 4-33.

*Example 4-33 Using correlation names*

---

```
$statement = "select c.id as c_id, d.id as d_id, c_email from customer c,
dealer d";
$stmt      = db2_prepare($conn,$statement);
$result    = db2_execute($stmt);

while ($row = db2_fetch_assoc($stmt)) {
    echo 'id is: ' . $row['C_ID'] . ' ' . $row['D_ID'] . ' email is: ' .
$row['C_EMAIL'];
}

if (!$result) {
    echo 'The execute failed. ';
    echo 'SQLSTATE value: ' . db2_stmt_error();
    echo 'with Message: ' . db2_stmt_errormsg();
}
```

---

- Use other methods of fetching, such as `db2_fetch_row`, `db2_fetch_array`, or `db2_fetch_both`, and use the column position indexes to get the values of the result set.

## Free 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.

```
// Connect to the database, use one of the following
// for PDO_IBM
$dbh = new PDO('ibm:dlrshp', '', '');
// for PDO_ODBC
$dbh = new PDO('odbc:dlrshp', '', '');

// Perform database operations here
...

// Free the connection
$dbh = null;
```

### 4.2.3 Unified ODBC

In this section, we describe the use of the Unified ODBC extension for DB2. The default dynamic scrollable cursor used to fetch data from the database is not supported by DB2, so DB2 downgrades the cursor to a dynamic keyset driven cursor. There is some performance impact.

In the case of a cataloged database, you can use the following command (where `dlrshp` is alias name for the cataloged database) to apply a CLI patch to automatically downgrade the cursor to a forward only cursor. Forward only cursor is faster than the scrollable cursor.

```
db2 UPDATE CLI CFG FOR SECTION dlrshp USING PATCH2 6
```

The Unified ODBC driver does not provide support for calling a stored procedure with the `OUT/INOUT` parameter. Because of this, it is better to use the `ibm_db2` extension of PHP to work with DB2.

Before using Unified ODBC, you need to configure parameters in `php.ini`:

- Large object handling

The default value of the parameter `odbc.defaultlrl` is 4 096 bytes. You should change this variable to a number of bytes the large object will hold. You can set this variable to a large number as shown below:

```
odbc.defaultlrl = 100000
```

- Handling persistent connections

You can enable, configure, and disable the persistent connection in Unified ODBC by changing parameters in `php.ini`:

```
odbc.allow_persistent = On
odbc.check_persistent = On
odbc.max_persistent = -1
odbc.max_links = -1
```

The `odbc.allow_persistent` parameter is used to allow or prevent persistent connections. Once you have enabled persistent connection, you can use `odbc.check_persistent` to make sure that the ODBC driver checks the links so that they are valid for reuse. You can control the maximum persistent connections made using `odbc.max_persistent` and the maximum number of connections made to the database by changing `odbc.max_links`. This includes both persistent and non-persistent connections. If these two parameters are set to -1, it means no limit as the maximum value.

► Handling binary data

You can force how the Unified ODBC driver handles binary data by setting different values for `odbc.defaultbinmode`. This variable can have three values:

- 0: Binary data is not handled (pass through).
- 1: Returns binary as binary.
- 2: Converts the binary to character and returns it.

The example for the default handling is given below.

```
odbc.defaultbinmode = 1
```

### ***Sample programs in DB2***

Once your DB2 installation is over, check out:

```
/home/db2inst1/sqllib/samples/php
```

for Unified ODBC-based samples which run on the sample database. The sample database can be created using `db2samp1` command.

### **Program flow**

The Unified ODBC program contains a set of ODBC functions which can be used for connecting and executing SQL statements against a DB2 database.

As in the case of `ibm_db2` programs, the program involves two types of resources:

► Connection resource

A connection resource is used to create a statement resource. It is obtained after the database is connected.

► Statement resource

The statement resource is used to execute the SQL and retrieve the result sets from the database.

The steps taken by a Unified ODBC program during transaction processing are:

1. Connect to the database.
2. Prepare and execute the statement.
3. Process the results.
4. Free the resources.

## **Connect to the database**

We can connect to both cataloged and non-cataloged databases using Unified ODBC driver. The Unified ODBC driver can have both persistent and non-persistent connections. The advantages of persistent connections are discussed in , “Connect to the database” on page 114. The functions `odbc_connect` and `odbc_pconnect` can be used for making persistent and non-persistent connections.

### ***Creating a connection to a cataloged database***

We can make connection to a cataloged database as shown in the below example by passing the database alias name, user name, and password to the `odbc_connect` or `odbc_pconnect` functions.

```
<?php
$database = 'd1rshp';
$user     = 'db2inst1';
$password = '123';
$conn     = odbc_connect($database, $user, $password);

if ($conn) {
    echo 'Connection succeeded.';
    odbc_close($conn);
} else {
    echo 'Connection failed.' . odbc_errormsg();
}
?>
```

### ***Creating a connection to a non-cataloged database***

In the following example, you can see connecting to a non-cataloged database by giving full connection details about the database in the connection string (DSN).

```
<?php
$database = 'd1rshp';
$user     = 'db2inst1';
$password = '123';
$database = "odbc:DSN={IBM DB2 ODBC
DRIVER};HOSTNAME=localhost;PORT=50001;DATABASE=d1rshp;PROTOCOL=TCPIP;UID=db
2inst1;PWD=123;";
$conn     = odbc_connect($database, '', '');
```

```

if ($conn) {
    echo 'Connection succeeded.';
    odbc_close($conn);
} else {
    echo 'Connection failed.' . odbc_errormsg();
}
?>

```

## Handling transactions

The Unified ODBC program runs on autocommit-enabled state. You can use `odbc_autocommit` to get the status of autocommit, whether it is ON or OFF. This function can also be used for resetting the autocommit status.

This function returns 1 when autocommit is enabled:

```
echo odbc_autocommit($conn);
```

To reset autocommit, pass the constant `false` along with it:

```
odbc_autocommit($conn, false);
```

Example 4-34 shows how to execute a transaction by explicitly doing a commit or rollback.

### *Example 4-34 Explicitly committing and rolling back a transaction*

---

```

<?php
$database = 'dlrshp';
$user     = 'db2inst1';
$password = '123';
$conn     = odbc_connect($database, $user, $password);

if ($conn) {
    echo 'Connection succeeded.';
    $statement = "INSERT INTO customer (c_name, c_email) VALUES ('Piotr',
'Piotr12@example.org')";

    odbc_autocommit($conn, false);

    // Prepare and execute the SQL statement
    $result = odbc_exec($conn, $statement);

    print_r(odbc_autocommit($conn));

    if (!$result) {
        odbc_rollback($conn);
        echo 'Execution failed: ' . odbc_errormsg();
    }
}

```

```

        odbccommit($conn);
        odbcautocommit($conn, false);
        odbcclose($conn);
    } else {
        echo 'Connection failed.' . odbcerrormsg();
    }
    ?>

```

---

## Execute the statement

In Unified ODBC, use the function `odbc_errormsg` to get the `SQLCODE` and message; use `odbc_error` to get the `SQLSTATE`.

The program should check the result every time an ODBC function is executed and if the result is false, display it, using:

```
echo 'ODBC function failed: ' . odbcerrormsg();
```

## Isolation level to use

Considerations of which isolation level to use are the same as for the `ibm_db2` extension. For details, refer to “Isolation level to use” on page 119. You can have the isolation level changed in the statement level using the **WITH** clause or you can change the `CURRENT ISOLATION` special register.

To execute an SQL in Unified ODBC, use `odbc_do` or `odbc_execute`. If the execute succeeds, it returns a result resource which can be used for fetching the result set. The following shows an example of using `odbc_do`.

```

$database = 'd1rshp';
$user     = 'db2inst1';
$password = '123';
$conn     = odbc_connect($database, $user, $password);
$statement = "select * from customer";

// Prepare and execute the SQL statement
/* odbc_exec($conn,$statement); // does the same as odbc_do */

$result = odbc_do($conn, $statement);

```

## Prepare and then execute

In the Unified ODBC extension, you can prepare the SQL statement with parameter markers and later execute it with parameters passed as an array. You can see in Example 4-35 how to do it.

*Example 4-35 Prepare and execute in Unified ODBC extension*

---

```
<?php
```

```

$database = 'dlrshp';
$user     = 'db2inst1';
$password = '123';
$conn     = odbc_connect($database, $user, $password);

if ($conn) {
    $statement = "INSERT INTO customer (c_name, c_email) VALUES (?,?) ";

    // Prepare the SQL statement
    $sth = odbc_prepare($conn, $statement);
    $email = '100@100.com';
    $name = 'Kiran';

    // Execute the SQL statement
    $result = odbc_execute($sth, array($name, $email));
    if (!$result) {
        odbc_rollback($conn);
        echo 'Execution failed: ' . odbc_errormsg();
    }

    odbc_close($conn);
} else {
    echo 'Connection failed.' . odbc_errormsg();
}
?>

```

---

## Process the results

The result set resource is returned after the SQL statement is executed using `odbc_do`, `odbc_exec`, or `odbc_execute`. The following Unified ODBC methods are available for retrieving the result set from the result set resource.

### ***Using odbc\_result\_all***

You can use the `odbc_result_all` function to display the fetched results in a table. The table properties can be given as the parameters for this function. See Example 4-36.

#### *Example 4-36 Using odbc\_result\_all to display the results*

---

```

$statement = "SELECT * FROM customer";

// Prepare and execute the SQL statement
$result = odbc_do($conn, $statement);

odbc_result_all($result, 'bgcolor="#eeeeee" border="1" width="30%");

if (!$result) {
    echo 'Execution failed: ' . odbc_errormsg();
}

```



```

}

odbc_free_result($result);

```

---

### ***Using odbc\_fetch\_row***

You can use the `odbc_fetch_row` function to retrieve the row. You can use the `odbc_result` to retrieve the column data. See Example 4-37.

#### ***Example 4-37 Using odbc\_fetch\_row***

```

$statement = "SELECT c_id, c_name, c_email FROM customer";

// Prepare and execute the SQL statement
$result = odbc_do($conn, $statement);

while (odbc_fetch_row($result)) {
    //collect results
    $id    = odbc_result($result, 1);
    $name  = odbc_result($result, 2);
    $email = odbc_result($result, 3);
    echo 'id is: ' . $id . ' and name is: ' . $name . ' and email is: ' .
$email;
}

if (!$result) {
    echo 'Execution failed: ' . odbc_errormsg();
}

odbc_free_result($result);

```

---

### ***Using odbc\_fetch\_array***

The function `odbc_fetch_array` can be used for fetching each row of the result set into an array indexed by the column names. An example is shown in Example 4-38.

#### ***Example 4-38 Using odbc\_fetch\_array to fetch the result set***

```

$statement = "SELECT c_id, c_name, c_email FROM customer";

// Prepare and execute the SQL statement
$result = odbc_do($conn, $statement);

while ($array = odbc_fetch_array($result)) {
    //collect results
    echo 'id is: ' . $array['C_ID'] . ' and name is: ' . $array['C_NAME'] . '
and email is: ' . $array['C_EMAIL'];
}

```

```

if (!$result) {
    echo 'Execution failed: ' . odbc_errormsg();
}

odbc_free_result($result);

```

---

### ***Using odbc\_fetch\_into***

You can use `odbc_fetch_into` to fetch a row into an array indexed by the respective column position. See Example 4-39.

#### *Example 4-39 Using odbc\_fetch\_into to fetch the result set*

---

```

$statement = "SELECT c_id, c_name, c_email FROM customer";

// Prepare and execute the SQL statement
$result = odbc_do($conn, $statement);

while (odbc_fetch_into($result,$array)) {
    //collect results
    echo 'id is: ' . $array[0] . ' and name is: ' . $array[1] . ' and email is: ' . $array[2];
}

if (!$result) {
    echo 'Execution failed: ' . odbc_errormsg();
}

odbc_free_result($result);

```

---

### ***Using odbc\_fetch\_object***

The function `odbc_fetch_object` returns an object whose properties are the column names of the fetched result set. See Example 4-40.

#### *Example 4-40 Using odbc\_fetch\_object to fetch result set*

---

```

$statement = "SELECT c_id, c_name, c_email FROM customer";

// Prepare and execute the SQL statement
$result = odbc_do($conn, $statement);

while ($object = odbc_fetch_object($result)) {
    //collect results
    echo 'id is: ' . $object->C_ID . ' and name is: ' . $object->C_NAME . ' and email is: ' . $object->C_EMAIL;
}

if (!$result) {

```

```

        echo 'Execution failed: ' . odbc_errormsg();
    }

    odbc_free_result($result);

```

---

### Free the resources

You can use `odbc_close` to free the connection resources and `odbc_free_result` to free the result resource. You can also do this by assigning these results to null value.

## 4.2.4 Porting PHP applications from Unified ODBC to ibm\_db2

If you want to use stored procedure, large objects, and to get better performance in a PHP program, you need to go for `ibm_db2` extension. It is very easy to port the unified ODBC application into `ibm_db2` extension code. These are the steps:

1. Change the function names.

Search and replace the string 'odbc\_' with 'db2\_'. Replace function `odbc_do` with `db2_exec`.

2. Change the error handling.

The error handling functions are different for `ibm_db2` since the connection error and the statement error are caught differently. Refer to “How to catch an error” on page 119 for error handling with `ibm_db2`.

For `ibm_db2`, the function names start with `db2_conn` for a connection specific error and `db2_stmt` for a statement specific error compared to the error reporting function of Unified ODBC extension which starts with `odbc_error`.

3. Change the function structure.

Most of the commonly used functions in Unified ODBC extension are the same as those in `ibm_db2` extension. Indexing the column starts with 0 in `ibm_db2` compared to 1 in Unified ODBC extension. You need to take care of this when using `db2_result`. If you want to get a row into an array while fetching, use `db2_fetch_array`, which returns an array with the values, compared to the `odbc_fetch_into` where the array is passed as a parameter.

4. Change the fetching style for better performance.

Use `db2_num_records` to find the number of records returned and then iterating that many times to get data using the fetch function. Use the return value from the fetch function to determine the exit condition for coming out of the fetch loop. If you want to find the number of rows returned, use SQL 'SELECT COUNT(\*)'.

Following is an example using the fetch function.

```
while ($row = db2_fetch_object($stmt)) {
    printf("%d %s %s", $row->C_ID, $row->C_NAME, $row->C_EMAIL);
}
```

## 4.3 Troubleshooting DB2 - PHP applications

All the PHP drivers for DB2 connection communicate with DB2 using Call Level Interface (CLI). CLI is a callable SQL interface written in C or C++ language. PHP extensions for DB2 are based on this interface. CLI is based on the ODBC (Open Database Connectivity) specification and CLI driver acts as an ODBC driver when loaded using ODBC driver manager.

CLI has an advanced tracing facility by which you can see the communication between your PHP application and the database. The details include the SQL statement, the values of parameters bound, the values fetched from the parameter, return code, and information about each and every CLI call made by the PHP program.

### 4.3.1 Taking CLI trace

The steps involved in taking CLI trace are:

1. Create a path to write the trace file.

Create a directory where you have given all the required permissions for every user to write trace into that directory.

On Windows:

```
mkdir c:\trace
```

On Linux/UNIX:

```
mkdir /home/db2inst1/trace
chmod 777 /home/db2inst1/trace
```

2. Update the CLI configuration.

You need to update the CLI configurations to log the trace. CLI configuration can be updated in two ways:

- a. Editing the db2cli.ini file.

You can find db2cli.ini file in C:\Program Files\IBM\SQLLIB in Windows and in /home/db2inst1/sqllib/cfg in UNIX.

```
[COMMON]
Trace=1
TracePathName=/home/db2inst1/trace
TraceComm=1
TraceFlush=1
```

If using Windows, change the TracePathName= c:\trace. After editing, leave a blank line at the end of the file.

b. Use command line.

You can use the DB2 command line to execute the following commands so that the CLI configuration gets updated.

```
db2 UPDATE CLI CFG FOR SECTION COMMON USING Trace 1
db2 UPDATE CLI CFG FOR SECTION COMMON USING TracePathName
/home/db2inst1/trace
db2 UPDATE CLI CFG FOR SECTION COMMON USING TraceComm 1
db2 UPDATE CLI CFG FOR SECTION COMMON USING TraceFlush 1
```

Using Windows, you have to change the TracePathName to windows path.

```
db2 UPDATE CLI CFG FOR SECTION COMMON USING TracePathName c:\trace
```

3. Confirm the updates in CLI configuration.

You use the following command to make sure that the updates made to the CLI configuration are taken by DB2.

```
db2 GET CLI CFG FOR SECTION COMMON
```

4. Restart the application.

The CLI configuration is read when the application starts.

5. Analyze the trace.

Once the application starts, you can see that log file gets created in the trace directory. You will have files created with p<processnumber>t<thread number>.cli

6. Stop tracing.

You can change the Trace=0 keyword in the [COMMON] section of the db2cli.ini or issue the following command:

```
db2 UPDATE CLI CFG FOR SECTION COMMON USING Trace 0
```

## Analyzing the trace file

We will analyze the CLI trace of the PHP program given in Example 4-41.

*Example 4-41 PHP program for CLI trace*

---

```
<?php
$database = 'dlrshp';
$user     = 'db2inst1';
$password = '123';
$conn     = db2_connect($database, $user, $password);

if ($conn) {
```

```

        $statement = "SELECT c_id, c_name, c_email FROM customer WHERE c_id > ? OR
c_name NOT LIKE ?" ;

        // Prepare the SQL statement
        $stmt = db2_prepare($conn,$statement);
        $id   = 100;
        $name = "Kiran";

        // Bind the parameters
        db2_bind_param($stmt, 1, "id", DB2_PARAM_IN);
        db2_bind_param($stmt, 2, "name", DB2_PARAM_IN);

        // Execute the SQL statement
        $result = db2_execute($stmt);

        while( $object = db2_fetch_object($stmt)) {
            //collect results
            echo 'id is: ' . $object->C_ID . ' and name is: ' . $object->C_NAME . '
and email is: ' . $object->C_EMAIL;
        }

        if (!$result) {
            echo 'The execute failed. ';
            echo 'SQLSTATE value: ' . db2_stmt_error();
            echo 'with Message: ' . db2_stmt_errormsg();
        }

        db2_close($conn);
    } else {
        echo 'Connection to database failed.';
        echo 'SQLSTATE value: ' . db2_conn_error();
        echo 'with Message: ' . db2_conn_errormsg();
    }
}
?>

```

---

You can see some sections of the CLI trace which are important when you analyze the trace.

The header in the information about the DB2 and the CLI Driver

```

[ Process: 21520, Thread: 1097019392 ]
[ Date & Time:          02/22/2006 21:07:34.175183 ]
[ Product:             QDB2/LINUX DB2 v8.1.2.97 ]
[ Level Identifier:    03040106 ]
[ CLI Driver Version:  08.01.0000 ]
[ Informational Tokens: "DB2
v8.1.2.97","special_15462","MI00142_15462","Fixpack 10"]

```

Once the program calls the db2\_connect, the PHP driver calls the SQLConnect.

```
SQLConnect( hDbc=0:1, szDSN="dlrshp", cbDSN=6, szUID="db2inst1", cbUID=8,
szAuthStr="***", cbAuthStr=3 )
----> Time elapsed - +1.320000E-004 seconds
sqlccsend( Handle - 1106008192 )
sqlccsend( ulBytes - 271 )
sqlccsend( ) rc - 0, time elapsed - +1.640000E-004
sqlccrecv( )
sqlccrecv( ulBytes - 124 ) - rc - 0, time elapsed - +1.400000E-005
sqlccsend( Handle - 1106008192 )
sqlccsend( ulBytes - 245 )
sqlccsend( ) rc - 0, time elapsed - +1.051500E-002
sqlccrecv( )
sqlccrecv( ulBytes - 291 ) - rc - 0, time elapsed - +7.611300E-002
( DBMS NAME="DB2/LINUX", Version="08.02.0003", Fixpack="0x23040106" )
( Application Codepage=1208, Database Codepage=1208, Char Send/Recv
Codepage=1208, Graphic Send/Recv Codepage=1200 )

SQLConnect( )
<--- SQL_SUCCESS Time elapsed - +1.851398E+000 seconds
( DSN="DLRSHP" )
( UID="db2inst1" )
( PWD="***" )
```

In the program we are preparing the SQL statement using db2\_prepare, which in turn calls the CLI function SQLPrepare. You can see from CLI trace clearly what SQL statement is executed and the parameter markers.

```
SQLPrepare( hStmt=1:1 )
----> Time elapsed - +1.330000E-004 seconds
( pszSqlStr="SELECT c_id, c_name,c_email FROM customer WHERE c_id > ? OR
c_name NOT LIKE ?", cbSqlStr=-3 )
( StmtOut="SELECT c_id, c_name,c_email FROM customer WHERE c_id > ? OR
c_name NOT LIKE ?" )

SQLPrepare( )
<--- SQL_SUCCESS Time elapsed - +1.606000E-003 seconds
```

Then we bind the parameter using db2\_bind\_param which calls the CLI function SQLBindParameter which binds a memory variable with the SQL query.

```
SQLBindParameter( hStmt=1:1, iPar=2, fParamType=SQL_PARAM_INPUT,
fCType=SQL_C_CHAR, fSQLType=SQL_VARCHAR, cbColDef=32672, ibScale=0,
rgbValue=&083fc02c, cbValueMax=5, pcbValue=&083fc4e4 )
----> Time elapsed - +1.010000E-004 seconds

SQLBindParameter( )
<--- SQL_SUCCESS Time elapsed - +4.100000E-004 seconds
```

Now after the bind, we call db2\_execute, which makes ibm\_db2 extension call CLI function SQLExecute. You can see the values of bound parameter while executing along with the CLI trace.

```
SQLExecute( hStmt=1:1 )
----> Time elapsed - +4.600000E-005 seconds
( Row=1, iPar=1, fCType=SQL_C_LONG, rgbValue=100 )
( Row=1, iPar=2, fCType=SQL_C_CHAR, rgbValue="Kiran" - x'4B6972616E',
pcbValue=-3, piIndicatorPtr=-3 )
  sqlccsend( Handle - 1106008192 )
  sqlccsend( ulBytes - 152 )
  sqlccsend( ) rc - 0, time elapsed - +6.000000E-006
  sqlccrecv( )
  sqlccrecv( ulBytes - 418 ) - rc - 0, time elapsed - +4.270000E-004
( Requested Cursor Attributes=3 )
( Reply Cursor Attributes=524298 )
( Actual Cursor Attributes=524299 )

SQLExecute( )
<--- SQL_SUCCESS Time elapsed - +1.706000E-003 second
```

Then, we fetch the row using db2\_fetch\_object. It actually involves a set of CLI function calls, but the important one to investigate is the SQLFetch function.

```
SQLFetch( hStmt=1:1 )
----> Time elapsed - +4.500000E-005 seconds
( iRow=1, iCol=1, fCType=SQL_C_DEFAULT, rgbValue=x'0A010000', pcbValue=4 )
( iRow=1, iCol=2, fCType=SQL_C_DEFAULT, rgbValue=x'50696F7472', pcbValue=5
)
( iRow=1, iCol=3, fCType=SQL_C_DEFAULT,
rgbValue=x'50696F74723132406578616D706C652E6F7267', pcbValue=19 )

SQLFetch( )
<--- SQL_SUCCESS Time elapsed - +8.210000E-004 second
```

You can get lots of information about the database activity that is happening when taking the CLI trace. It is the most useful tool when debugging DB2 specific problems in PHP.

### 4.3.2 db2diag.log

Whenever you find that there is a problem with the database manager or the database, you should look into db2diag.log. You can find this file in ~/sqllib/db2dump directory in Linux/UNIX and C:\Program Files\IBM\SQLLIB\DB2 in Windows by default.



The level of logging is decided by the DIAGLEVEL database manager configuration parameter and the default is 3. You can have a maximum value of 4 for DIAGLEVEL. You can update the diagnostic level using the command:

```
db2 UPDATE DBM CFG USING DIAGLEVEL 4
```

Example 4-42 is a sample error message when we tried to export from a table which does not exist.

*Example 4-42 db2diag.log entry*

---

```
2006-02-24-20.50.06.873000-480 I104731H429          LEVEL: Severe
PID      : 5420                                TID   : 2748          PROC  : db2bp.exe
INSTANCE: DB2                                NODE  : 000
APPID    : *LOCAL.DB2.060225045005
FUNCTION: <0>, <0>, <0>, probe:877
MESSAGE  : SQL0204N "ADMINISTRATOR.PHOTOS" is an undefined name.
DATA #1  : String, 12 bytes
sqluexpm.SQC
DATA #2  : String, 44 bytes
Error from Import/Export or Load Processing
```

---

### 4.3.3 Tools for monitoring, tuning, and troubleshooting

DB2 ships with many easy to use troubleshooting and monitoring tools that can help you pinpoint where problems occur. This section provides an overview of some of these tools.

#### ***db2pd***

Use the **db2pd** tool to analyze the database and database manager. This tool can be used to see the details about database transactions, table spaces, table statistics, dynamic SQL, database configurations, buffer pools, locks, table spaces, and many other database-related details. A single db2pd command can retrieve multiple areas of information and can route the output to files. You can invoke db2pd a specified number of times within a specified period of time using the repeat parameter, and this will help you to determine changes over time. You can use this tool for problem determination, database monitoring, troubleshooting, and performance tuning. It is similar to the onstat command in Informix. This tool is executed from the command line with optional interactive mode. You can see an example of the use of db2pd to get the agent information in Example 4-43.

*Example 4-43 Using db2pd*

---

```
db2pd -agents
```

Agents:

Current agents: 2  
Idle agents: 0  
Active agents: 2  
Coordinator agents: 2

Address	AppHandl	[nod-index]	AgentTid	Priority	Type	State
ClientPid	Userid	ClientNm	Rowsread	Rowswrtn	LkTmOt	DBName
0x0257C040	8	[000-00008]	1016	0	Coord	Inst-Active 0
db2event	11	3	NotSet	DLRSHP		
0x012B30F0	7	[000-00007]	2284	0	Coord	Inst-Active 3740
ADMINIST	db2bp.ex	16	0	NotSet	DLRSHP	

---

### ***DB2 Snapshot***

This tool helps you to get the status of the database system. The monitoring is started by monitor switches in the instance and application level. The information is about buffer pool, locks, SQL statement, sorting, table activity, timestamp, and unit of work. The monitoring can be enabled at both instance level and application level. You can use the following command to enable it in instance and application level:

- ▶ **UPDATE DBM CONFIGURATION** command (instance level)
- ▶ **UPDATE MONITOR SWITCHES** command (application level)

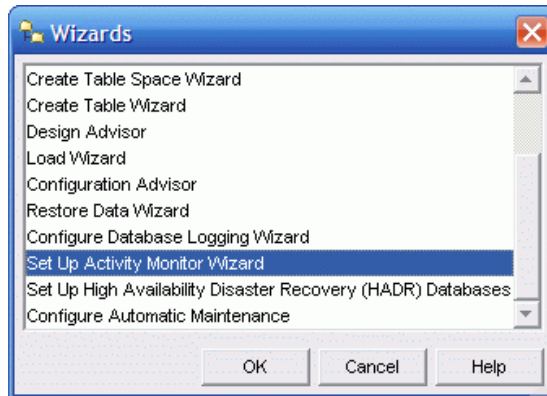
Use the **GET SNAPSHOT** command with the respective parameters to get the snapshot information. You can get this information inside the application using the SQL statement below. This SQL calls the table functions which return this information.

```
SELECT * FROM TABLE( SNAPSHOT_LOCK ('d1rshp',-1)) AS snapshot_lock
```

In addition to LOCK, there are other types of snapshot information available. These functions return tables which could be queried using SQL.

### **DB2 activity monitor**

You can start DB2 Activity Monitor Wizard from **Control Center** → **Tools** → **Wizards**. See Figure 4-1.



*Figure 4-1 Selecting your wizard*

You can use this graphical wizard to select a monitoring task for the following reasons (Figure 4-2):

- ▶ Resolving general database slow down
- ▶ Resolving performance degradation of the application
- ▶ Resolving an application locking situation
- ▶ Tuning dynamic SQL statement cache

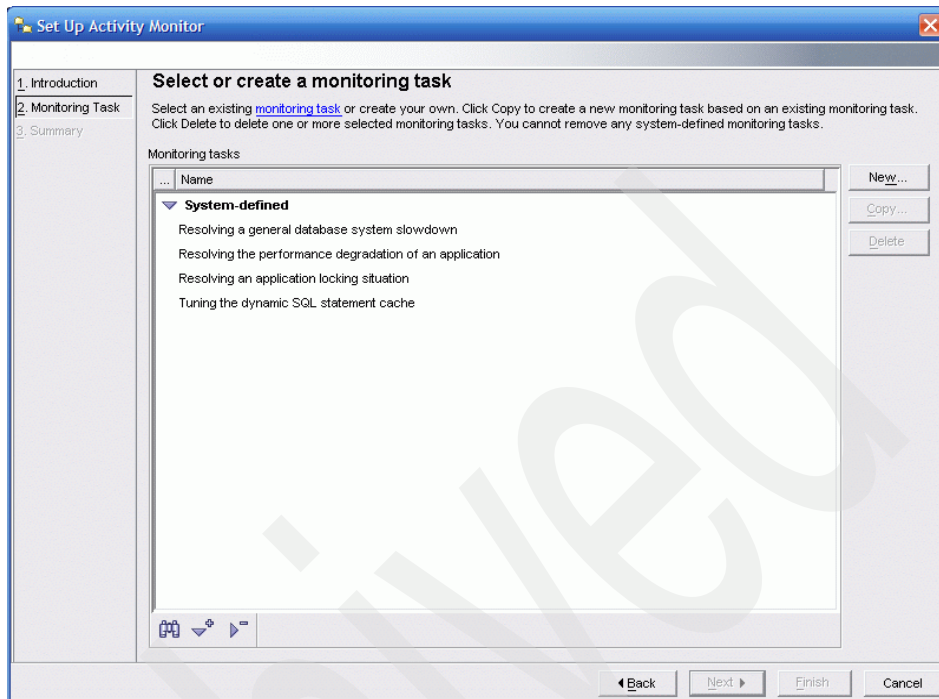


Figure 4-2 Create monitoring task

### **DB2 Event monitor**

Event monitors are database objects used to collect information when specified events occur or changes occur. There is no switch, but you need to create the event monitors for events you need to monitor. Even monitors are associated with events, such as database events, table events, deadlocks, table spaces, connections, buffer pools, and transactions.

We use the `CREATE EVENT MONITOR` command to create an event monitor. You can write the event monitor data to a table pipe or a file.

You can also create the event monitor from the Control Center (Figure 4-3).

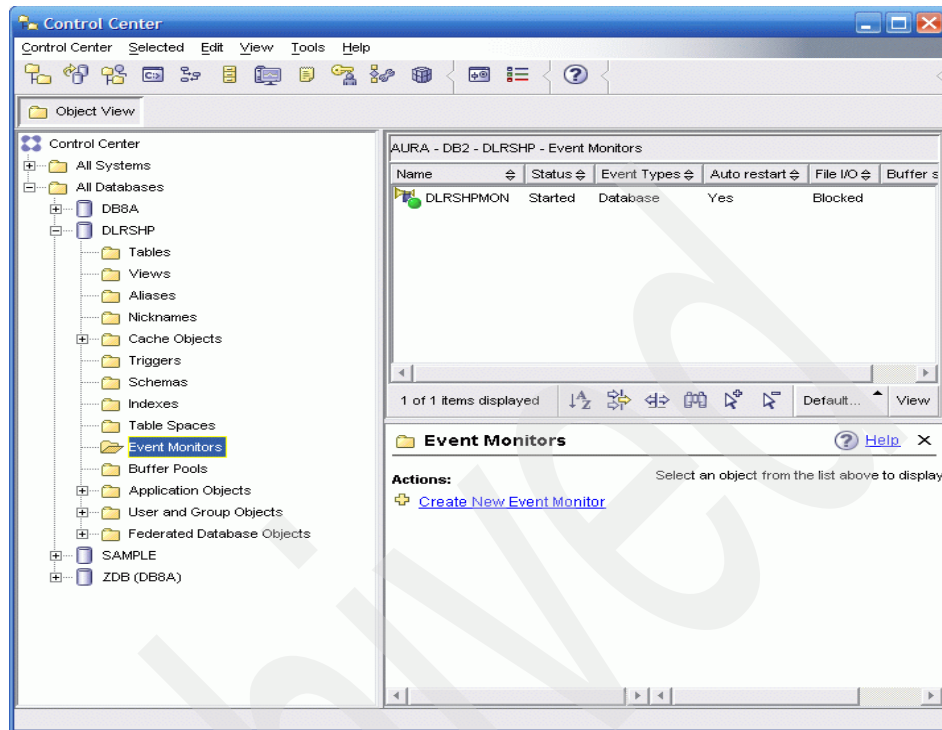


Figure 4-3 Event monitors in Control Center

You can review the events for the event monitor using Event analyzer. You can get this by right-clicking the event monitor entry in the Control Center. You can create the event monitor using the Create Event Monitor wizard. See Figure 4-4.

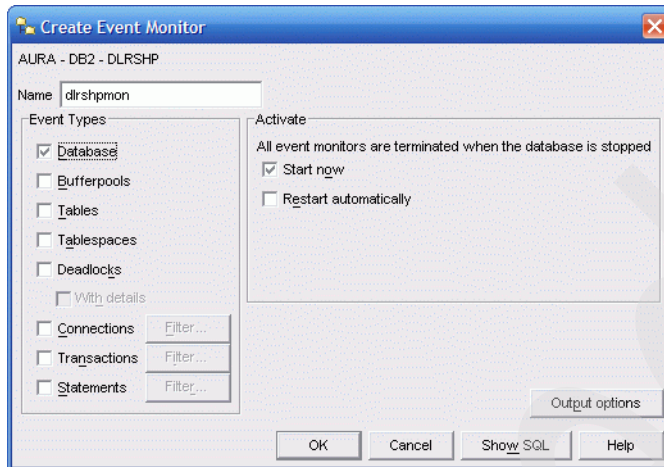


Figure 4-4 Create Event Monitor

Once you have created the event monitors, right-click the event monitor entry in the Control Center to create an instance of event analyzer (Figure 4-5). You can use this utility to view the details of the information gathered by the event analyzer.

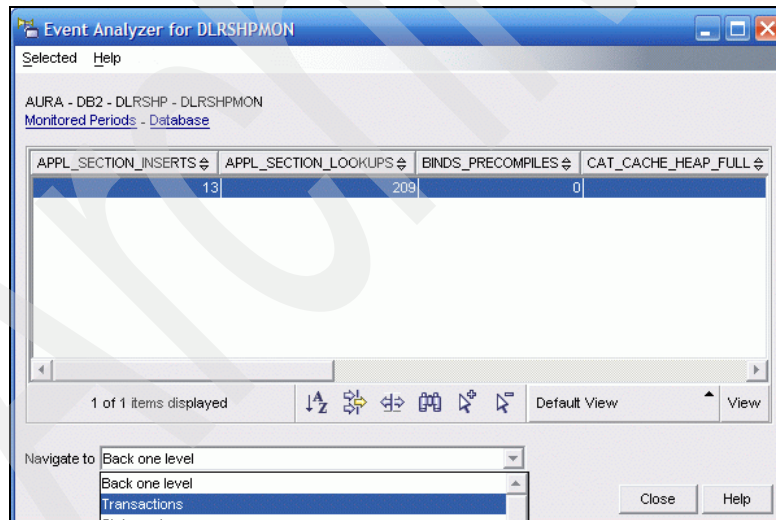


Figure 4-5 Event Analyzer

### **DB2 Explain facility**

Use the DB2 explain facility to see the access plan for the SQL and all related data which DB2 optimizer used to make a decision determining the access plan.

Do the following steps to use the explain facility.

1. Create the explain tables (the DDL is in ~/sql11b/misc/EXPLAIN.DDL).
2. Populate the explain table before running the PHP program, use the SQL command SET EXPLAIN MODE or SET EXPLAIN SNAPSHOT.
3. Use tools like db2exfmt or db2expln to format the explain table data and retrieve the access plan and other information.

You can use the graphical visual explain tool in the Control Center to see the access plan and the information used to decide the access plan by the optimizer, which includes cardinality of the table, CPU speed, available memory, I/O, and many others.

Figure 4-6 shows the access plan of SQL in Example 4-11 on page 127.

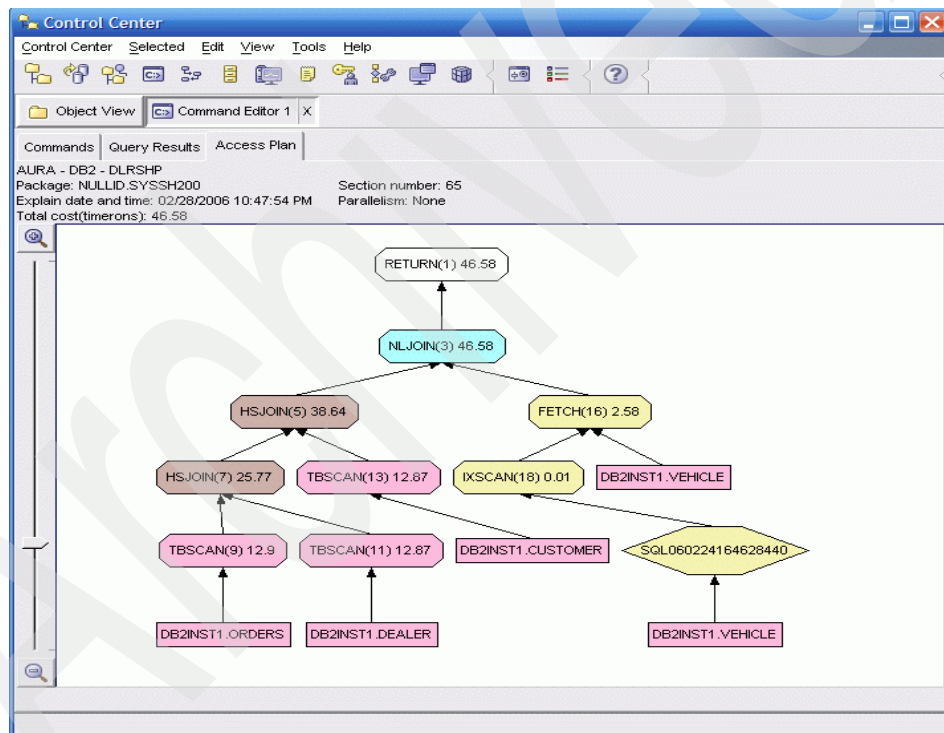


Figure 4-6 Access plan

### Using DB2 design advisor

You can use db2advls to find out the recommendations for creating database objects based on an existing database and a set of queries. In Example 4-44,

you can see how db2advis can be used to recommend indexes for an SQL from the ORDERS and CUSTOMERS tables from our sample application.

*Example 4-44 db2advis*

```
db2advis -db dlrshp -n db2inst1 -type I -s "select c_id,c_name,o_id, f_v_id
from db2inst1.customer, db2inst1.orders where c_id = f_c_id"
```

The output is

execution started at timestamp 2006-02-28-23.47.40.882000

Recommending indexes...

total disk space needed for initial set [ 0.018] MB

total disk space constrained to [ 4.100] MB

Trying variations of the solution set.

Optimization finished.

2 indexes in current solution

[ 26.0000] timerons (without recommendations)

[ 0.0818] timerons (with current solution)

[99.69%] improvement

--

--

-- LIST OF RECOMMENDED INDEXES

-- =====

-- index[1], 0.009MB

CREATE UNIQUE INDEX "DB2INST1"."IDX603010748040000" ON "DB2INST1"."CUSTOMER"  
("C\_ID" ASC) INCLUDE ("C\_NAME") ALLOW REVERSE SCANS ;

COMMIT WORK ;

RUNSTATS ON TABLE "DB2INST1"."CUSTOMER" FOR INDEX  
"DB2INST1"."IDX603010748040  
000" ;

COMMIT WORK ;

-- index[2], 0.009MB

CREATE INDEX "DB2INST1"."IDX603010748050000" ON "DB2INST1"."ORDERS"  
("F\_C\_ID"

ASC, "F\_V\_ID" ASC, "O\_ID" ASC) ALLOW REVERSE SCANS ;

COMMIT WORK ;

RUNSTATS ON TABLE "DB2INST1"."ORDERS" FOR INDEX  
"DB2INST1"."IDX60301074805000  
0" ;

COMMIT WORK ;

--

--

-- RECOMMENDED EXISTING INDEXES

-- =====

--

--

-- UNUSED EXISTING INDEXES

-- =====

-- =====



--

10 solutions were evaluated by the advisor  
DB2 Workload Performance Advisor tool is finished.

---

From the example above, you can see that db2advis has recommended some indexes. You need to create the database objects recommended by db2advis for better performance for your application.

You can also use the DB2 Design Advisor wizard from **Control Center** → **Tools** → **Wizards**. See Figure 4-7.

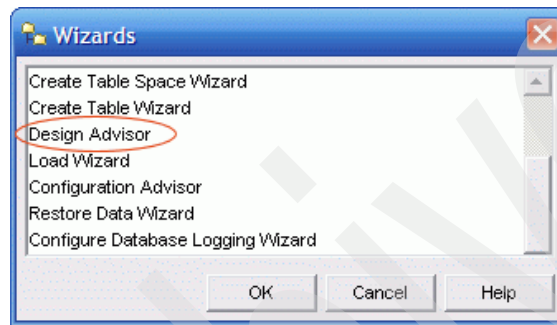


Figure 4-7 DB2 Design Advisor

#### 4.3.4 Getting the best out of DB2 from PHP

Once your PHP application is up and running, you may want to try to tune both DB2 and your application to get the best performance out of them. DB2 has a world class cost-based optimizer to make sure that each query you write gets executed in the best manner possible depending on your environment. DB2 can handle terabytes of data and is used by many enterprises for the best performance possible.

##### ***Design considerations for faster PHP-DB2 programs***

For better performing applications using DB2 and PHP, we need to take care from the design phase of the project. The following points need to be considered while you design and start coding your application.

- ▶ Use the prepare and then execute method for executing SQL statements which are repeatedly executed.
- ▶ Try to use connection pooling in PDO driver and persistent connection in ibm\_db2 driver or enable connection concentrator in DB2.
- ▶ Commit often, make sure that locks do not hold for a long time.

- ▶ Try to push more logic to the database layer and use user defined functions, store procedures, and SQL/XML, and, therefore, reduce network traffic and save computing resources.
- ▶ Make use of all your disks, store your tables, which are large, in table spaces spread over different disks, and store DB2 logs in different disks.
- ▶ Index your table columns appropriately.(Use DB2 Design Advisor).
- ▶ Change your database and database manager configuration to optimize db2 for your environment (Use Configuration Advisor wizard for making changes).
- ▶ Make use of stored procedures and compound SQL to reduce network traffic.
- ▶ Do 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.

## 4.4 PHP application and DB2 for z/OS

You can develop a PHP application on Linux or the Windows environment to access the data in z/OS or iSeries DB2 databases. To facilitate the connection for the DB2 Client on Linux, UNIX, or Windows to DB2 for z/OS database, DB2 Connect is required. DB2 Connect is included in DB2 ESE or can be installed stand along. The DB2 Connect is a licensing product.

Once the z/OS database is cataloged, there is no difference in developing a PHP application with DB2 database on Linux, UNIX, or Windows than with DB2 for z/OS. A PHP driver has been written using CLI (Call Level Interface) which takes care of all requirements needed to connect to z/OS machines. An existing PHP program can run against DB2 for z/OS without any change as long as the table structures are the same.

The information needed to catalog DB2 database on z/OS are:

- ▶ IP address
- ▶ Port number
- ▶ The location name of DB2 subsystem
- ▶ User name and password

Request these details from the mainframe database administrator. In our scenario, we are connecting to an IP address of 9.12.6.70 and port number of 12345. The DB2 subsystem is db8a.

The steps involved in cataloging the database are

### ***Catalog the node***

The following command catalogs the node and names the node z/390:

```
db2 CATALOG TCPIP NODE z390 REMOTE 9.12.6.70 SERVER 12345
```

We need to give it a name which we will use to refer to the database in the z/390 node and this name is given arbitrarily.

```
db2 CATALOG DATABASE zd1rshp AT NODE z390 AUTHENTICATION DCS
```

### ***Catalog the database***

We need to make our cataloged database name refer to the location name in the DB2 mainframe subsystem.

```
db2 CATALOG DCS DATABASE zd1rshp AS db8a
```

We can verify the connection by using the `CONNECT TO` statement and by providing user name and password.

```
db2 CONNECT TO zd1rshp USER <username> USING <password>
```



## PHP applications with Informix database servers

This Chapter discusses in-depth the installation and configuration of the application development environment with Informix IDS as the database server. It provides a survey about available PHP extensions for developing PHP applications with Informix IDS and can give you the ability to discover their functional strength in order to use them in your environment.

The first section gives an global overview about how to install and configure Informix IDS, Apache Web server, PHP, and several PHP database extensions in order to bring these components to work together. The second section, a key part of this chapter, describes in detail the usage of the programming interfaces provided by the selected PHP database extensions through the use of various examples. We discuss the following SQL and database concepts:

- ▶ Database connections
- ▶ Dynamic and static statements
- ▶ Cursors and result sets
- ▶ Stored procedures
- ▶ Transaction management and isolation level
- ▶ Complex data types
- ▶ BLOBs and SBLOBs
- ▶ Error handling

## 5.1 Application environment setup

The application environment planning includes, depending on the time of the project and the budget, a production and a development environment. Most of the time, these two environments do not share the same resources. Generally, these two environments are different in terms of sizing and configuration, but they have the same major influencing factors:

- ▶ Software
- ▶ Hardware

### Software

The starting point for software planning today is the decision to use open source products or commercial software products. This is not only valid for the operating system, it can also be considered for all components needed by the application environment. Once this specific decision is made, naturally completely different products stay on your selection list.

In our specific project environment, PHP is the selected application language and the Informix IDS is the database server used for this chapter. Quite open is the decision for the operating system and the Web server.

You must be aware of the different requirements for the development and production environments from several standpoints. Web server and PHP extensions for the database come in different versions and interfaces which have to be selected depending on strength, functionality, and needs in the project. In the development environment, there is often a coexistence of versions and products. There are additional requirements for a development, Quality Assurance (QA), and debugging environment which are not necessary on the production environment.

### Hardware

The other major factor for the application environment setup is the hardware component. Using a UNIX RISC environment requires a different hardware base than Intel® Linux or Windows. The sizing requirements for a production environment with 10 000 parallel users and a database larger than 1 TB will be different from the system running an application with five parallel users and 10 GBs of data.

But, today sizing is not the only influencing factor for hardware. Working in a Web environment requires spending much more effort to establish security. Most of the time, the database server and Web server are separated into different machines. This is also favorable from the security point of view. The Web server is used as a service accessible from the outside, meaning the Internet. This will

require that the Web server machine is located in the part of a company's infrastructure which has the highest security. In opposition, a database server has no needs to serve requests from the Web directly, this hardware can be situated behind the firewall in the intranet.

When planning the development environment, the requirements to focus on security are not as high as they are for the production environment. The database server and Web server could be configured on one machine in the intranet, but this infrastructure requires higher efforts for QA simulating the production environment.

### 5.1.1 Lab application environment

In this redbook, we set up the application environment as follows:

- ▶ The Web server, database server, and application clients are on separate machines.
- ▶ The application clients are based on Windows XP, and Internet Explorer is the Web browser.
- ▶ The Web servers were installed on a Red Hat RHEL4 on an IBM xSeries machine. We tested both Apache Web server Versions 1.3.34 and 2.0.55 compiled from the source and the Apache package shipped with the Red Hat distribution. In addition, we set up PHP 5.1.2 with the various Informix connectivity packages on this machine. To make sure that PHP was able to access the database, IBM Informix Client SDK 2.90.UC3 was installed.
- ▶ For the database server, we used IBM Informix IDS V10.00.UC3R1 on SUSE Linux SLES9.

In the following sections, we discuss the setup of several components for this project in detail.

### 5.1.2 Installing IBM Informix IDS V10.00 on Linux

This section describes in detail how to install the Informix IDS on Linux. Following that, the configuration and initialization of a single database instance is discussed briefly. For details and complete configuration settings, refer to the Informix library at:

<http://www.ibm.com/software/data/informix/pubs/library/index.html>

#### Requirements

IBM Informix IDS database server is a very scalable database engine. The memory and disk space requirements are strongly related to the specifications of projects running on the server. Here we list some basic installation requirements.

### **Supported Linux distributions by IBM Informix IDS**

In general, IBM Informix IDS V10.00 runs and is supported on the SUSE SLES 9 and Red Hat RHEL 3/4 releases for kernel 2.6. A complete list of supported Linux distributions and hardware platforms can be found at:

<http://www.ibm.com/software/data/informix/linux/ids.html>

### **Supported hardware by IBM Informix IDS**

IBM Informix IDS supports the following hardware:

- ▶ x86 (32-bit edition for Intel Pentium®, Xeon®, and AMD Athlon-based systems)
- ▶ POWER (IBM eServer™, System i (iSeries), and System p™ (pSeries®))
- ▶ zSeries® (IBM System z™)
- ▶ Intel EM64T (X86-64), which is available with the first IBM Informix Dynamic Server Version 10.0 Fix pack (Q2 2005)
- ▶ IA64 (64-bit edition for Intel Itanium®-based systems)
- ▶ AMD64 (64-bit edition for AMD Opteron and Athlon64-based systems), which is available with the first IBM Informix Dynamic Server Version 10.00 Fix pack

### **Users**

Before starting IBM Informix IDS installation, you need to create a new user and a new group. The name of the user and group must be *informix*. The user *informix* must be in the group *informix*. Create the group first and after that, the user.

From the shell, the way to create a new group and a new user is similar for SUSE and Red Hat. See below for the usage of `groupadd` and `useradd`:

```
groupadd informix
useradd informix -m -g informix -d /home/informix
```

User root can create a new password with:

```
passwd informix
```

Another way to generalize the creation of users with password in one shell command is the `-p` option in `useradd`. The parameter after the `-p` is not the password itself but is the encrypted password string. A simple C program with a `crypt` library call to generate the encrypted password is shown in Example 5-1:

#### **Example 5-1 Generating a password for `useradd -p`**

---

```
#define _XOPEN_SOURCE
#include <unistd.h>
#include <stdio.h>
```



```

main(int argc, char **argv)
{
    char * result;
    extern char * crypt();

    if ( argc!=2 )
    {
        printf("Usage : crypt <passwordstring>");
        exit(1);
    }
    result =crypt(argv[1], "az");
    printf("crypt returns : %s \n", result ? result : "");
}

```

---

The password character string is the input and it returns the encryption. For example:

```

crypt 123456
Returns: azG9m1zRG10fo

```

This string could be added with the `-p` option of `useradd` to create user and password in one step.

The new user and the group also could be added to the system via a graphical interface.

In SLES9, use Yast, Choose **Security and Users** → **Edit and Create Groups** to add a new Group. A new User can be created with **Security and Users** → **Edit and Create Users**.

In Red Hat, you can do this operation via the Applications menu in the Users X Interface, Choose **Applications** → **System Settings** → **User and Groups**.

### ***Disk space***

IBM Informix IDS and IBM Informix Client SDK bundle for Linux require about 400 MB. Additional space is needed for additional products, database, and data.

### **Installation**

For Linux, IBM Informix IDS V10 can be acquired by various means. There is a 30 day trial version for download at the Web site:

[http://www.ibm.com/developerworks/downloads/im/ids/?S\\_TACT=105AGX28&S\\_CMP=DLMAIN](http://www.ibm.com/developerworks/downloads/im/ids/?S_TACT=105AGX28&S_CMP=DLMAIN)

In addition, an express edition is available for Linux and Windows environments.

IBM Informix IDS provides great flexibility in product installation. There is no restriction on where the product has to be installed, so one machine can have multiple installations of one version, or several versions.

To keep the description of IBM Informix IDS installation simple, we assume the following:

- ▶ The installation media file is `iif10.00.UC3R1.LINUX-I32.zip`.
- ▶ The installation directory is `/sqldists/1000UC3`.

Following are the steps to install IBM Informix IDS V10 Linux:

- ▶ Log in into Linux as user root.
- ▶ Create installation directory:  
`mkdir /sqldists/1000UC3`
- ▶ Change the directory ownership to the informix user:  
`chown informix:informix /sqldists/1000UC3`
- ▶ Copy and change the current directory:  
`cp iif10.00.UC3R1.LINUX-I32.zip /sqldists/1000UC3`
- ▶ Unzip and untar the file:  
`gunzip iif10.00.UC3R1.LINUX-I32.zip`  
`tar -xvf iif10.00.UC3R1.LINUX-I32.tar`
- ▶ Set up the `INFORMIXDIR` environment variable:  
`export INFORMIXDIR=/sqldists/1000UC3`
- ▶ Run the install utility:  
`./install_ids -gui`

The complete installation procedure is described in the manual *Getting Started*, G251-2758-02, available at:

<http://publib.boulder.ibm.com/infocenter/idshelp/v10/index.jsp>

## Configuration

After the successful installation of IDS, there are a few additional steps for setting up a database instance. The successful setup of the instance is required for enabling the access for PHP and the Web server. The instance of the database server will maintain all database objects needed by the Web applications, such as tables, views, permissions, and so on.

### */etc/services*

The installation was completed by user root. After that, we start enabling the communication between the Web server and the database server. Both

machines need to know on which port the database server serves all incoming requests. Add a new entry in the `/etc/services` file on the machine where the database server is installed. You need to specify here an unused port and the communication protocol. Example 5-2 shows you a sample entry.

*Example 5-2 Modify the `/etc/services` file for definition of the communication port*

---

```
ids1000uc3 22222/tcp
```

---

### **`$INFORMIXDIR/etc/sqlhosts`**

The `sqlhosts` file, from the database server view, specifies which kind of communication will be used for the instance. In a new installation, this file does not exist. You have to create this file or use the template provided under `$INFORMIXDIR/etc/sqlhosts.std`. To use the template, copy the `sqlhosts.std` as `sqlhosts`. In Example 5-3, we show a sample entry for a communication over the network with sockets commonly used for Informix IDS under Linux. The comment in the first line explains the position parameters in the entry.

*Example 5-3 Sample entry for network communication in `$INFORMIXDIR/etc/sqlhosts`*

---

```
#<database server name> <communication> <hostname> <service name/port>
on1000UC3soc          onsoctcp      Linux        ids1000uc3
```

---

The database server name is the instance name, which we are configuring. This name must match an entry in the `DBSERVERNAME` or `DBSERVERALIASES` configuration parameter.

`onsoctcp` described the server communication interface. This communication type must be used within Linux when the applications and the database server are on different machines. If the database server and the Web server are on one machine and both of them are compiled with same bit, 32 or 64 bit, `onipcshm` could be alternatively used.

The host name is the UNIX server name where the IDS was installed.

The service name is the name specified in the `/etc/services` file, one step before. The entry in the `/etc/services` is not required if a port number is used as long as the service name in `sqlhosts` and the port number are not used by another application.

### **`ONCONFIG`**

The major configuration file for each database server is the `onconfig` file. The name could be changed, but must be distinct for each server instance. One machine can have multiple Informix IDS versions and one IDS server can have multiple instances. Each instance must have its own `onconfig` file. All of them must reside in the `$INFORMIXDIR/etc` directory. An additional discussion about

how to distinguish between different instances is in “Environment variables” on page 195. There is a skeleton file `onconfig.std` in the `$INFORMIXDIR/etc` directory. This can be copied and modified. The original `onconfig.std` must be kept for the server to look up standard settings if variables are undefined in the instance configuration file.

The goal in this discussion is the basic setup for an instance which is working. There are a lot of performance parameters which could be changed. But to get the instance up and running, most of them could be left as their initial default values. All variables in the configuration file, their settings, and meanings are described in the documentation *Administrator's Guide for Informix Dynamic Server*, G251-2267-02.

Starting from the beginning of the file, you have to change first the `ROOTPATH` and the `ROOTSIZE` configuration parameters. The `ROOTPATH` to a cooked file or a raw device is initially used for all necessary elements, such as physical log files, databases, and tables. In comparison with DB2, the closest term can be container. `ROOTPATH` has to point to an existing file owned by `informix` with the group `informix` and the permissions of `660`. Using a cooked file, the file has to be created with `touch` and can have a size of `0`. The database server will initialize the file to the specified size. When you use a raw device, make sure that the configured size in `ROOTSIZE` matches the size of the raw device. If using a raw device created by a logical volume manager, we recommend not to start the `ROOTOFFSET` configuration parameter at offset `0` but to leave the first `32 KB` free. Sometimes, the first `32 KBs` are used by the volume manager to write administration information. If there are collisions between Informix data and volume manager data, it could lead to data loss or data corruptions.

The `ROOTSIZE` also depends on the `PHYSSIZE`, `LOGSIZE`, and `LOGFILES` configuration parameters. The `PHYSSIZE` defines, together with `PHYSDBS` configuration parameter, the size and location of the before images needed for recovery purposes. The `LOGSIZE` and `LOGFILES` define the number and the size of logical logs of the instance. These parameters are the most critical parameters in the system. Depending on the number of parallel users and the average size of transactions per user, these parameters should be adjusted properly.

For the lab environment, the following settings were chosen:

```
onstat -c | grep ROOT
ROOTNAME      rootdbs          # Root dbspace name
ROOTPATH      /sqldists/1000UC3/chunks/root_chunk
ROOTOFFSET    0                # Offset of root dbspace into device (Kbytes)
ROOTSIZE      500000           # Siz of root dbspace (Kbytes)

onstat -c | grep PHYS
PHYSDBS       rootdbs          # Location (dbspace) of physical log
```

PHYSFILE            10000            # Physical log file size (Kbytes)

onstat -c | grep LOG

LOGFILES            10            # Number of logical log files

LOGSIZE            2000            # Logical log size (Kbytes)

Other important parameters that should be changed are:

SERVERNUM           0            # Unique id corresponding to a OnLine instance

DBSERVERNAME on1000UC3soc # Name of default database server

If you have more than one database server instance on the machine, you need to give each of them a distinct number specified by the SERVERNUM configuration parameter. With this number, the server will generate a unique key when creating shared memory segments at system startup. The server name is the name of the instance and has to match one entry in the sqlhosts file.

After adjusting these parameters, a couple of parameters related to file accessing should be updated. The MSGPATH configuration parameter specifies the log file where all server messages are written. This should point to an existing file. The path in the ALARMPROGRAM configuration parameter should be changed to the installation directory.

### ***Environment variables***

There are only a few environment variables you need to update:

- ▶ INFORMIXDIR: This variable points to the installation directory.  
**export INFORMIXDIR=/sqldists/1000UC3**
- ▶ ONCONFIG: This variable contains the name of the configuration file for the specific database instance. The configuration file must be located in the etc directory pointed to by the INFORMIXDIR environment variable.  
**export ONCONFIG=onconfig**
- ▶ PATH: PATH should include the bin directory of the Informix installation directory.  
**export**  
**PATH=/usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:/usr/games:/opt/gnome/bin:/opt/kde3/bin:/usr/lib/java/jre/bin:/home/informix/v10/bin**
- ▶ INFORMIXSERVER: This variable specifies which database server is used. The value should match either the DBSERVERNAME or DBSERVERALIASES configuration parameter in the onconfig file and the database server name specified in the sqlhosts file.  
**export INFORMIXSERVER=on1000UC3soc**

## Initialization

After setting up the files and the environment variables for the instance, the server can be initialized. This will be done by informix user with the following command at the shell prompt:

**oninit -iy**

The **-i** option is only issued the first time for initialization, all subsequent attempts to bring up the instance should be done by only issuing the:

**oninit**

command from the Shell.

Once the oninit comes back to shell without any error, check the successful initialization in the log file. The messages, such as Example 5-4, indicate the initialization was successful.

### *Example 5-4 Server log file after successful initialization*

---

```
IBM Informix Dynamic Server Version 10.00.UC3R1  -- On-Line -- Up 00:1:48 --
86636 Kbytes

10:25:33 IBM Informix Dynamic Server Version 10.00.UC3R1  Software Serial
Number AAA#B000000
10:25:48 IBM Informix Dynamic Server Initialized -- Complete Disk Initialized.
10:25:48 Checkpoint Completed: duration was 0 seconds.
10:25:48 Checkpoint loguniq 1, logpos 0xd0, timestamp: 0x4f

10:25:48 Maximum server connections 0
10:25:48 Dataskip is now OFF for all dbspaces
10:25:48 On-Line Mode
10:25:48 Building 'sysmaster' database ...
10:25:51 Logical Log 1 Complete, timestamp: 0x634a.
10:25:52 Booting Language <spl> from module <>
10:25:52 Loading Module <SPLNULL>
10:25:54 Logical Log 2 Complete, timestamp: 0x9157.
10:25:54 Unloading Module <SPLNULL>
10:25:57 Loading Module <SPLNULL>
10:25:58 'sysmaster' database built successfully.
10:25:58 'sysutils' database built successfully.
10:25:58 'sysuser' database built successfully.
```

---

Now, the database server is able to handle new connection requests. It is time to implement the data model into the database, setting up all necessary database objects, such as tables, views, stored procedures, and initially load the necessary data.

## Troubleshooting at initialization time

We have compiled a list of problems commonly seen during Informix IDS initialization. The solution for the problem is included in the example.

- ▶ Rootchunk does not exist.

Example 5-5 shows the error messages displayed on the window after issuing `oninit -iy`.

*Example 5-5 Rootchunk does not exist*

---

```
oninit -iy
oninit: Cannot open chunk '/home/informix/v10/chunks/root_chunk1'. errno= 2
oninit: Fatal error in shared memory initialization
```

---

Action:

- Make sure that the file defined in the ROOTCHUNK configuration parameter really exists.

- ▶ Insufficient permissions.

The `oninit` returns no error message. However, verifying server state using `onstat -` does not show that the server came up successfully as seen in Example 5-6.

*Example 5-6 Verify server state*

---

```
onstat -
shared memory not initialized for INFORMIXSERVER 'on1000UC3soc'
```

---

Run `onstat -m` to figure out the reason. It shows that the permissions on the rootchunk are not correct. See Example 5-7.

*Example 5-7 Rootchunk does not have sufficient permissions*

---

```
onstat -m
02:45:47 DR: DRAUTO is 0 (Off)
02:45:47 Dynamically allocated new virtual shared memory segment (size
8192KB)
02:45:47 IBM Informix Dynamic Server Version 10.00.UC3R1 Software Serial
Number AAA#B000000
02:45:47 The chunk '/home/informix/v10/chunks/root_chunk' must have
READ/WRITE permissions for owner and group (660).
```

---

Action:

- Change the permissions for the rootchunk to 660.

- ▶ Rootchunk size is too small.

Several configuration parameter settings in the `onconfig` file can influence the size of the rootchunk. Example 5-8 shows a common error message indicating that the configured size for the chunk is too small.

*Example 5-8 Rootchunk size is too small*

---

```
oninit -iy
oninit: Not enough room in ROOT DBspace.
        Requested 31038K, ONCONFIG value 'ROOTSIZE' 5000K.
```

---

Action:

- As discussed in the section about the `ONCONFIG` parameters, Informix IDS will try to allocate the space for logs as specified in `LOGFILES` and `PHYSSIZE` during the initialization time. Make sure that the rootchunk is large enough to contain all the necessary structure. A good starting point is the requested size in the error message.

► Missing environment variables setting.

Another important step before initializing the server is the appropriate setting of the environment variables. Example 5-9 shows an error of missing the setting of the variable `$INFORMIXSERVER`.

*Example 5-9 INFORMIXSERVER environment variable is wrong at initialization*

---

```
oninit -iy
INFORMIXSERVER does not match DBSERVERNAME or any of the DBSERVERALIASES.
FAILED
```

---

Action:

- Review the setting of the `INFORMIXSERVER` environment variable. This setting has to match with the `DBSERVERNAME` parameter in the configuration file and in the `sqlhosts` entry.

### 5.1.3 Installing the Apache Web server

After the successful installation, configuration, and initialization of the database server, the next step is to install and configure the Web server. This section covers the build, installation, and configuration of Apache V1.3 and Apache V2.0 in detail. If there are differences between Red Hat and SUSE, we discuss it separately.

We separate the Web server and the database server in different machines. On the Web server machine, we installed Apache, PHP, and Informix client software. Before starting the installation, you have to decide if you want to use the precompiled Apache packages shipped with the Linux distribution, or build your own Web server distribution.



The benefit of using precompiled packages is saving time. There is no need for setting up the source tree and building the distribution. The packages are maintained by the RPM database in Linux.

Downloading and setting up the source tree give the Web administrator the flexibility to compile the source with the option specially needed by the application. Also the Web administrator is open for choosing a specific stable Apache version.

Another benefit for using your own distribution is the installation of the complete Web server software, including the executable and the configuration files, in one installation directory. The packages from the Linux distributions tend to spread the files across the file systems.

## Installing the Apache shipped with the Linux distribution

This section provides a brief overview of using the Web server from the various Linux distributions, including which versions are available, how to verify the installation of the packages, and how to install the Web servers in case they are not already installed with the default Linux installation.

### *Red Hat*

By default, Apache Web server 2.0 will be installed when installing RHEL4. We can verify this by issuing the command:

```
rpm -qa | grep http
httpd-manual-2.0.52-12.ent
httpd-2.0.52-12.ent
httpd-suexec-2.0.52-12.ent
system-config-httpd-1.3.1-1
```

The following shows the output of a quick package version verification:

```
rpm -qi httpd-2.0.52-12.ent
Name       : httpd                      Relocations: (not relocatable)
Version    : 2.0.52                    Vendor: Red Hat, Inc.
Release    : 12.ent                    Build Date: Mon 28 Feb 2005
04:26:16 AM PST
Install Date: Thu 26 Jan 2006 07:42:22 PM PST    Build Host:
porky.build.redhat.com
Group      : System Environment/Daemons    Source RPM:
httpd-2.0.52-12.ent.src.rpm
Size       : 2413331                    License: Apache Software
License
Signature  : DSA/SHA1, Mon 16 May 2005 09:29:27 AM PDT, Key ID
219180cddb42a60e
Packager   : Red Hat, Inc. <http://bugzilla.redhat.com/bugzilla>
```

```
URL      : http://httpd.apache.org/
Summary  : The httpd Web server
```

Only Apache V2.0 is available in this Red Hat version. Apache V1.3 must be built and installed manually if needed. There is no option to install this from the X Session via **System Settings → Add Remove Application → Web Server**.

Since the installation directory of the binary package is not `/usr/local/apache2`, a quick way to verify where the configuration file was installed is using `rpm` command:

```
rpm -ql httpd-2.0.52-12.ent | grep httpd.conf
/etc/httpd/conf
/etc/httpd/conf.d
/etc/httpd/conf.d/README
/etc/httpd/conf.d/welcome.conf
/etc/httpd/conf/httpd.conf
/etc/httpd/conf/magic
```

## **SUSE**

SLES 9 does not install the Apache Web server by default. This must be manually done from:

### **yast → install and Remove Software**

Search for “apache”.

In SLES 9, there are both Apache Web servers V1.3.29 and V2.0.49 available. With SUSE 10, only Apache V2 is shipped. A verification for an installed Apache package with SUSE can be done by:

```
rpm -qa | grep apache
apache2-example-pages-2.0.49-27.8
apache2-prefork-2.0.49-27.8
apache2-mod_php4-4.3.4-43.8
apache2-2.0.49-27.8
```

Make sure the example pages for Apache are installed. With the default configuration, if they are not installed, you will see an access denied window in the browser when you try to start the Web server. In difference with the installation of Apache from a manual build, Apache V2 has to be started with the `apache2ctl` command. For Apache V1.3, there is no difference, you can use the `apachectl` script for the startup. The reason is that SLES9 provides a possible installation of both Apache versions and you have to distinguish the startup scripts in the system.

## Build and install Apache Web server from the source code tree

This section discusses the steps for setting up the Apache Web server from the source code tree. There are no differences between Red Hat and SUSE for the build and installation processes.

Note that we only discuss the general steps to compile the Apache Web server. These steps are sufficient for a development environment. For the production environments, additional options should be considered for system security purposes. For further discussion about how to set up SSL (Secure Socket Layer) with Apache Web server and its different modules, refer to:

<http://apache-ssl.org>

<http://modssl.org>

### **Apache V2.0**

To set up Apache V2.0, follow these steps:

1. Download the source.

The source code is available at the Web site:

<http://httpd.apache.org/>

2. Configure and build.

Copy the downloaded file to the build directory and run the following commands:

```
gunzip httpd-2.0.55.tar.gz
tar -xvf httpd-2.0.55.tar
cd httpd-2.0.55
configure --enable-so
make
```

3. Install Apache V2.0, run:

```
make install
```

The Apache distribution is copied to /usr/local/apache2.

### **Apache V1.3**

To install Apache V1.3:

1. Download the source.

The source code for Apache Version 1.3 is also available at:

<http://httpd.apache.org/>

2. Configure and build.

Copy the download file into the build directory and run the following commands:

```
gunzip apache_1.3.34.tar.gz
tar -xvf apache_1.3.34.tar
cd apache_1.3.34
./configure --prefix=/usr/local/apache1 --enable-module=so
make
```

- Install Apache V1.3, run:

```
make install
The Apache distribution is copied to /usr/local/apache1.
```

Building from the source tree gives you the opportunity to specify your own distribution path. It is very useful in the development environment where multiple server installations are required on one machine. We also used this in our lab environment for comparison purposes.

### Configuration of the Apache Web server

After the successful installation of the Web server, there are a few changes necessary in the `httpd.conf` file. The `httpd.conf` file for the Apache distribution built from source tree could be found in the installation directory (for the given example, assume `/usr/local/apache2/conf`). For the Web server installed from Red Hat, use the following steps to get the location of the file:

```
rpm -qa | grep http
httpd-2.0.52-12.ent
rpm -ql httpd-2.0.52-12.ent | grep httpd.conf
```

For the Web server installed from SLES9, use the following rpm statements to determine the location of the file:

```
rpm -qa | grep apache
apache2-2.0.49-27.8
rpm -ql apache2-2.0.49-27.8 | grep httpd.conf
```

Comparing with the installation from the source, the difference is that the package files are not necessarily in one directory tree.

The following parameters in the `httpd.conf` should be checked and changed:

- ServerName
- DocumentRoot
- LoadModule
- AddType
- Directory

## 5.1.4 Installing the Informix client and connectivity

Depending on the application environment architecture, Informix Connect or Informix SDK is the next item to be installed. If the Web server and database

server are on one machine and Informix SDK was already installed with Informix IDS in the bundle, there is no need to install the client product.

If two machines and the `ifx_*` function set for the PHP database interface are to be used, the installation of Informix SDK is required to get PHP compiled. For PDO and unixODBC, only Informix Connect is required to provide the connectivity to the database server at run time.

Following are the required installation steps:

1. Create group and user *informix*.
2. Create installation directory.
3. Set the `INFORMIXDIR` environment to the installation directory.
4. Unpack installation media into the installation directory.
5. Run the install command:
  - For SDK: Run `installclientsdk -gui`
  - For Connect: Run `installconn -gui`

After the installation of the product, in case you use a separate machine as the Web server, you must copy the `sqlhosts` file from the database server machine to the `etc` subdirectory pointed to by the `INFORMIXDIR` environment variable in the Web Server machine. If you use an entry in the `/etc/services` on the database server, a similar entry has to exist on the Web server machine.

### 5.1.5 Setting up PHP with multiple Informix IDS interfaces

The Informix IDS database server supports a number of PHP database interface extensions, including the interfaces based on the native ESQL/C, CLI, and ODBC. In addition, the Informix PDO extension supporting PHP PDO was announced recently. This database interface is completely based on the object-oriented capabilities newly introduced in PHP 5. The PEAR DB is a hybrid implementation provides the application a object-oriented programming interface. Internally, PEAR DB is based on the procedural ESQL/C interface layer.

This section describes in-depth the setup, build, and installation of four different interfaces, enabling the PHP application the access to Informix IDS databases.

- **PDO:** As an object-oriented interface to support the efforts making PHP code more independent from the underlying database server. It has been available since PHP Version 5.1.
- **ifx\_\* function set:** As a procedural interface for Informix IDS enabling the access since PHP Version 3.

- ▶ **PEAR DB:** As an object-oriented PEAR module, based on the `ifx_*` function set with the goal to make the code more portable (similar to PDO). It has been available since PHP Version 5.0.
- ▶ **unixODBC:** A procedural interface based on the ODBC standard. This interface uses a third party ODBC manager. It has been available since PHP Version 3.

There are other interfaces which can enable access to Informix, but they are not discussed in this redbook. To find all the available interfaces and the details, refer to the following Web sites:

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

All interfaces in the following discussion are set up with PHP 5.1.2.

## PHP, Informix, and Linux distributions

Similar to the Web server, you have the option of using PHP with a precompiled library from the Linux distribution, or building your own libraries. Using the binaries from Linux distribution would save you the effort of setting up, building, and installing the source tree but the choice of interfaces and versions is also limited.

But be careful, do not mix the self-build libraries with the libraries that come with the Linux distribution. Most of the time the versions are incompatible. Keep in mind that the base PHP version for RHEL4 and SLES9 is Version 4, your build will mostly be based on Version 5, so the load of the shared extension will probably fail with undefined symbol errors. Example 5-10 shows the mismatch of an ODBC-shared extension taken from Version 4, but the PHP executable was based on Version 5.

### *Example 5-10 Version mismatch between PHP and a shared extension*

---

```
php pdoconnect0.php
PHP Warning:  PHP Startup: Unable to load dynamic library
'/usr/lib/php4/odbc.so' - /usr/lib/php4/odbc.so: undefined symbol: empty_string
in Unknown on line 0
```

---

## **RHEL4**

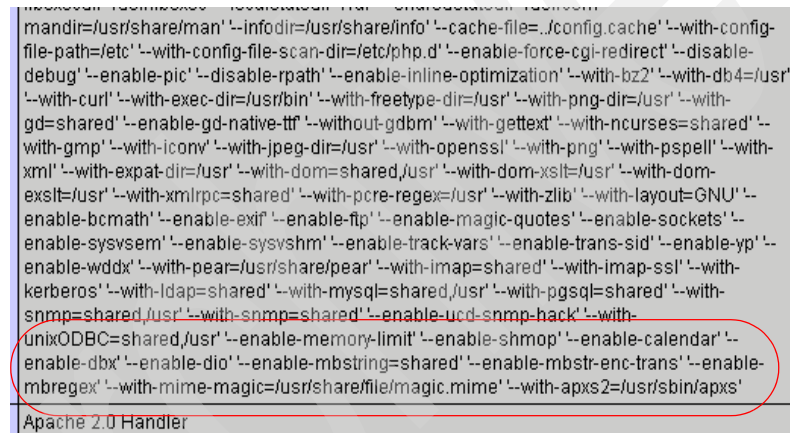
In a default installation, the RHEL4 distribution will install PHP Version 4.3.9 and some shared extensions. This PHP is also tied together with the Apache V2.0 package installed by default. Since the extension could be linked in PHP statically, you need to check the availability of the database interfaces in the PHP base package. Since this is PHP Version 4, the interfaces we look for will not be the Informix PDO. Using the `phpinfo` script shown in Example 5-11 is an easy

way to check which options are built into the shipped PHP package. If you have manually compiled packages and multiple PHP libraries on different Web server versions, this script can help you figure out the actual configuration. This script should be in the `ROOTDOCUMENT` directory specified by the `httpd.conf` file on the Apache server.

*Example 5-11 Using `phpinfo()` to get the PHP configuration in the Web server*

```
<?php
phpinfo();
>
```

Calling this simple PHP script from a browser will generate a HTML output listing a table with several sections. We are interested here in the configuration options which show us the interfaces statically included. Figure 5-1 shows the PHP library shipped with the RHEL4 distribution.



```
mandir=/usr/share/man' '--infodir=/usr/share/info' '--cache-file=../config.cache' '--with-config-
file-path=/etc' '--with-config-file-scan-dir=/etc/php.d' '--enable-force-cgi-redirect' '--disab-
le-debug' '--enable-pic' '--disable-rpath' '--enable-inline-optimization' '--with-bz2' '--with-db4=/usr'
'--with-curl' '--with-exec-dir=/usr/bin' '--with-freetype-dir=/usr' '--with-png-dir=/usr' '--with-
gd=shared' '--enable-gd-native-ttf' '--without-gdcm' '--with-gettext' '--with-ncurses=shared' '--
with-gmp' '--with-iconv' '--with-jpeg-dir=/usr' '--with-openssl' '--with-png' '--with-pspell' '--with-
xml' '--with-expat-dir=/usr' '--with-dom=shared,/usr' '--with-dom-xslt=/usr' '--with-dom-
exslt=/usr' '--with-xmlrpc=shared' '--with-pcre-regex=/usr' '--with-zlib' '--with-layout=GNU' '--
enable-bcmath' '--enable-exif' '--enable-ftp' '--enable-magic-quotes' '--enable-sockets' '--
enable-sysvsem' '--enable-sysvshm' '--enable-track-vars' '--enable-trans-sid' '--enable-yp' '--
enable-wddx' '--with-pear=/usr/share/pear' '--with-imap=shared' '--with-imap-ssl' '--with-
kerberos' '--with-ldap=shared' '--with-mysql=shared,/usr' '--with-pgsql=shared' '--with-
snmp=shared,/usr' '--with-snmpp=shared' '--enable-ucd-snmpp-hack' '--with-
unixODBC=shared,/usr' '--enable-memory-limit' '--enable-shmop' '--enable-calendar' '--
enable-dbx' '--enable-dio' '--enable-mbstring=shared' '--enable-mbstr-enc-trans' '--enable-
mbregex' '--with-mime-magic=/usr/share/file/magic.mime' '--with-apxs2=/usr/sbin/apxs'
```

Apache 2.0 Handler

*Figure 5-1 Compile options for PHP from the Red Hat distribution*

The configuration parameter list of the `unixODBC` interface is the one to which we paid attention. There is a small problem attempting to using the `unixODBC` with the library provided by the standard RHEL4 distribution. If you use the small test program taken from Example 5-27 on page 225, you will receive a message “unknown function `odbc_connect`”. The reason is that the `unixODBC` extension library was built as shared, as indicated by the prefix *shared*. The extension has to be installed separately to include two additional packages and an ODBC manager which will handle the connection requests. The names of the two additional packages are:

- ▶ `unixODBC-2.2.9-1` for the ODBC Manager
- ▶ `php-odbc-4.3.9-3.6` for the PHP `unixODBC` extension

There are additional setup steps of configuring the unixODBC with the Informix IDS required. These detailed steps about how to set up the ODBC and the PHP to work with Informix IDS are described in “The unixODBC interface” on page 213.

Here is a tip about how to find out the reason for “unknown function” messages in PHP. In the RHEL4 distribution, the libraries and the PHP executable are stripped so nm command will not find any of them defined. But with phpinfo the configuration can easily be captured. Building the PHP source with the same configuration string can then be done. To find out if the function is included in the build, run the following command in the build directory:

```
nm sapi/cli/php | grep <your function>
```

Example 5-12 shows that the Red Hat configuration string in a separate build will return nothing from nm php | grep odbc because the shared library is not included.

*Example 5-12 Using nm to get defined symbols from libraries and executable*

```
> pwd
/home/php/php-5.1.2.source
>nm sapi/cli/php | grep odbc_connect
080d5a30 T zif_odbc_connect
```

**SUSE SLES9**

With SLES9, you can install the PHP base in Version 4.3.4. It is not installed by default. There are extensions using shared libraries which could be installed separately. Installing a PHP extension from the installation media requires the PHP base package named php4-4.3.4-43.8 in SLES 9. Necessary changes in the Apache configuration file httpd.conf needed by PHP will be automatically done at installation time.

Check availability of the PHP interfaces supported by Informix IDS. We do this in the same way as described for the RHEL4. You can use the same PHP script presented in Example 5-11 on page 205. The output of the script is shown in Figure 5-2.

Configure Command	./configure '--prefix=/usr' '--datadir=/usr/share/php' '--mandir=/usr/share/man' '--bindir=/usr/bin' '--libdir=/usr/share' '--includedir=/usr/include' '--sysconfdir=/etc' '--with-_lib=lib' '--with-config-file-path=/etc' '--with-exec-dir=/usr/lib/php/bin' '--disable-debug' '--enable-inline-optimization' '--enable-memory-limit' '--enable-magic-quotes' '--enable-safe-mode' '--enable-sigchild' '--disable-ctype' '--disable-session' '--without-mysql' '--disable-cli' '--without-pear' '--with-openssl' '--with-apxs2=/usr/sbin/apxs2-prefork' '1586-suse-linux'
-------------------	---

*Figure 5-2 Configure options for the SLES9-based PHP library*



You will notice that there is no database interface statically built in. Similar to the RHEL4, SLES9 only provides a shared extension library for unixODBC. You have to install unixODBC manually. Two additional extensions and the unixODBC manager package also should be installed. Any necessary configuration changes for the PHP with the Apache Web server will be done automatically. The names of the additional extension packages are:

- ▶ unixODBC-2.2.8-58.4
- ▶ php4-unixODBC-4.3.4-43.8

There are additional setup steps for configuring the unixODBC with the Informix IDS that should be done. For a detailed description about how to set up the ODBC and the PHP to work with Informix IDS, refer to “The unixODBC interface” on page 213.

**Note:** If you decide to install the PHP from the Linux distribution on a Linux system where you have already installed a Zend product, `/etc/php.ini` is a link to the `php.ini` file in the Zend directory. This `php.ini` file contains an extension directory in the Zend section for the extensions provided by Zend. When the PHP base package is installed, the installer adds another directory entry in the general section for the additional extensions. The PHP executable will take the last occurrence of the extension directory for determining the extension libraries. This will be the Zend directory. If you try to execute a script with functionality related to the new installed extension, this will fail with “unknown function” messages. To make sure that the extension libraries from the Linux distribution will be used, copy the new PHP libraries to the Zend extension directory.

## Building the PHP with Informix from the source tree

Using the default binaries in the Linux distributions would restrict you in selecting the PHP version and database interface usages. To reach much more flexibility and get access to the new object-oriented capability of PHP 5.1, you have to set up and build the PHP with the appropriate interface for your application. You also can combine all the interfaces into one PHP library.

We describe the setup and build of each of the interfaces separately. We focus on building statically the interfaces into the PHP. For Informix PDO, we have an additional discussion about how to build the shared extension. The method can be applied to the other interfaces. Before starting to describe in detail the setup specifics for each interface, following are a few steps needed for every interface setup:

1. Download the PHP source from:

<http://www.php.net/downloads.php>

2. Set up build directory:  

```
gunzip php-5.1.2.tar.gz
tar -xvf php-5.1.2.tar
```
3. Verify the directory setup:  

```
./configure --help
```

### ***Include the Informix PDO statically***

Informix PDO is an extension for PHP which provides a set of database objects maintaining the database connection, statements, and result sets. It is available with PHP Version 5.1. The Informix PDO is not included in the downloaded PHP source code tree. Before compiling PHP, you need to download and configure Informix PDO:

1. Download the Informix PDO to /tmp from the following Web site:  
[http://pecl.php.net/package/PDO\\_INFORMIX](http://pecl.php.net/package/PDO_INFORMIX)
2. Unzip the file:  

```
gunzip PDO_INFORMIX-1.0.0.tgz
tar -xvf PDO_INFORMIX-1.0.0.tar
```
3. Move the files to the directory <PHP sourcecode>/ext/  

```
mv PDO_INFORMIX-1.0.0 <PHP sourcecode>/ext/pdo_informix
```
4. Include the Informix PDO in the configure script:  

```
buildconf --force
```
5. Verify that the new configuration includes the Informix PDO:  

```
configure --help | grep informix-pdo
--with-pdo-informix=DIR Include PDO Informix support. DIR is the base.
```
6. Configure PHP for using Informix PDO and Apache:
  - For Apache V2.0  

```
configure --with-pdo-informix=/home/informix/sdk
--with-apxs2=/usr/local/apache2/bin/apxs --with-config-file-path=/etc
```
  - For Apache V1.3  

```
configure --with-pdo-informix=/home/informix/sdk
--with-apxs=/usr/local/apache1/bin/apxs --with-config-file-path=/etc
```
7. Build and install the PHP.  

```
make
make install
```

After executing all these steps, the PHP library is installed under:  
 /usr/local/apache2/module for Apache V2 or /usr/local/apache1/libexec for Apache V1.3. The PHP and PEAR command line utilities are installed in

/usr/local/bin. The PEAR repository can be found in /usr/local/lib/php. If you use the Apache Web server from the Linux distribution, and you want to build the PHP manually, the parameters --with-apxs or --with-apxs2 should point to your Apache installation directory.

There are additional steps necessary for configuring PHP with Apache. These steps are described in “Configuring Apache with PHP” on page 215.

### ***Include the Informix PDO as shared***

We discuss now the inclusion of the Informix PDO into the PHP as a shared library. This can be taken as an example and could also applied to the ifx\_\* and the unixODBC interface. Do the following steps:

1. Build the PHP with Informix PDO statically included.  
Follow the steps described in “Include the Informix PDO statically” on page 208.
2. Find the shared library directory.

During building the PHP with Informix PDO, the shared library *pdo\_informix.so* is also installed in the `make install` step. The shared library installation directory is shown in the `make install` output as shown in Example 5-13.

#### ***Example 5-13 make install output for a shared Informix PDO and PHP***

---

```
[root@lead php-5.1.2.pdo_shared]# make install
Installing PHP SAPI module:      apache2handler
/usr/local/apache2/build/instdso.sh
SH_LIBTOOL='/usr/local/apache2/build/libtool' libphp5.1a
/usr/local/apache2/modules
/usr/local/apache2/build/libtool --mode=install cp libphp5.1a
/usr/local/apache2/modules/
cp .libs/libphp5.so /usr/local/apache2/modules/libphp5.so
cp .libs/libphp5.lai /usr/local/apache2/modules/libphp5.1a
libtool: install: warning: remember to run `libtool --finish
/home/holgerk/php/php-5.1.2.pdo_shared/libs'
chmod 755 /usr/local/apache2/modules/libphp5.so
[activating module `php5' in /usr/local/apache2/conf/httpd.conf]
Installing PHP CLI binary:      /usr/local/bin/
Installing PHP CLI man page:    /usr/local/man/man1/
Installing shared extensions:
/usr/local/lib/php/extensions/no-debug-non-zts-20050922/
Installing build environment:    /usr/local/lib/php/build/
Installing header files:         /usr/local/include/php/
Installing helper programs:      /usr/local/bin/
    program: phpize
    program: php-config
Installing man pages:            /usr/local/man/man1/
```

```

page: phpize.1
page: php-config.1
Installing PEAR environment:      /usr/local/lib/php/
[PEAR] Archive_Tar      - already installed: 1.3.1
[PEAR] Console_Getopt  - already installed: 1.2
[PEAR] PEAR             - already installed: 1.4.6
Wrote PEAR system config file at: /usr/local/etc/pear.conf
You may want to add: /usr/local/lib/php to your php.ini include_path
Installing PDO headers:          /usr/local/include/php/ext/pdo/

```

---

► Add the Informix PDO shared extension to php.ini.

In order to get the shared library to work with the PHP, add your `extension_dir` and extension to the `/etc/php.ini` (if they are not already there) as shown in Example 5-14.

*Example 5-14 Add PHP shared extension to php.ini*

---

```

; Directory in which the loadable extensions (modules) reside.
extension_dir = /usr/lib/php4
extension = pdo_informix.so

```

---

There are steps necessary to configure the PHP with Apache. These steps are described in “Configuring Apache with PHP” on page 215.

### ***The ifx\_\* function set***

The `ifx_*` function set has been available in the PHP since Version 3. It contains a set of ESQL/C files and uses embedded statements. The setup prerequisite is that the Informix SDK package has been installed. The following steps are necessary for configuration, build, and installation in order to run PHP with Apache:

1. Set up the environment variable to the Informix SDK installation directory:

```
export INFORMIXDIR=/home/informix/sdk
```

2. Configure the PHP with Informix and Apache.

– For Apache V2.0, run:

```
configure --with-informix=/home/informix/sdk
--with-apxs2=/usr/local/apache2/bin/apxs --with-config-file-path=/etc
```

– For Apache V1.3, run:

```
configure --with-informix=/home/informix/sdk
--with-apxs=/usr/local/apache1/bin/apxs --with-config-file-path=/etc
```

If the following text is seen, the configuration was successful:

```

+-----+
| License:                                     |

```

This software is subject to the PHP License, available in this distribution in the file LICENSE. By continuing this installation process, you are bound by the terms of this license agreement. If you do not agree with the terms of this license, you must abort the installation process at this point.

Thank you for using PHP.

### 3. Build and install the PHP with ifx\_\* functions.

```
make
make install
```

After executing these steps, the PHP library is installed in: `/usr/local/apache2/module` for Apache V2 or `/usr/local/apache1/libexec` for Apache V1.3. The PHP command line utility and PEAR are installed in `/usr/local/bin`. The PEAR repository is in `/usr/local/lib/php`. If you use the Apache Web server from the Linux distribution and you want to build the PHP manually, the parameters `--with-apxs` or `--with-apxs2` should point to your Apache installation directory.

There are additional steps required for configuring PHP with Apache. These steps are described in “Configuring Apache with PHP” on page 215.

## Troubleshooting

The Informix SDK is the prerequisite for building PHP with the `ifx_*` function set. You need to have the environment variables set. The environment variable `INFORMIXDIR` has to be set before the configure step. Example 5-15 shows the error message you will get without the setting.

### Example 5-15 Missing INFORMIXDIR when executing configure

**configure ...**

```
.....
checking if your cpp allows macro usage in include lines... yes
checking for IMAP support... no
checking for IMAP Kerberos support... no
checking for IMAP SSL support... no
checking for Informix support... yes
configure: error: INFORMIXDIR environment variable is not set.
```

If you have an older Linux, such as SUSE Professional 9.1 or partially upgraded Linux distributions, some packages may be missing for configuring or building the product. The name of the package and the minimum version number will be shown in the error message.

### ***The PEAR DB package***

PEAR is an extension and application repository. It provides a very useful set of libraries for tasks like authentication, database access, XML, and other services to PHP application development.

The DB package is a PEAR extension. It provides an unified database interface code regardless of the database used behind the application. It contains a set of PHP files, which describe objects and object methods for using the databases. The unification of the database interface certainly limits the ability of the database server to support some specific function.

The PEAR DB is available in the RHEL4 and SLES9 distributions. But with this package alone, you will not be able to access an Informix IDS database. The DB package, in view of Informix connectivity, needs a PHP library with the `ifx_*` functions configured. The manual installation of PHP with the `ifx_*` functions is discussed in “The `ifx_*` function set” on page 210.

The following steps are necessary for DB package installation. Make sure that PEAR is already installed, either within the Linux distribution or with a previous PHP installation.

1. Verify if PEAR DB is already installed:

```
pear list
Installed packages, channel pear.php.net:
=====
Package      Version State
Archive_Tar  1.3.1   stable
Console_Getopt 1.2     stable
PEAR         1.4.6   stable
```

2. Download the package from:

<http://pear.php.net/package/DB/download>

3. Unzip the file.

```
gunzip DB-1.7.6.tgz
```

4. Install PEAR DB.

```
pear install DB-1.7.6.tar
```

5. Verify the Installation of PEAR DB.

```
pear list
Installed packages, channel pear.php.net:
=====
Package      Version State
Archive_Tar  1.3.1   stable
Console_Getopt 1.2     stable
DB          1.7.6   stable
```

### ***The unixODBC interface***

Using unixODBC, the PHP client connects over the standard ODBC interface to the database server. To use unixODBC, you need to set up a data source via an external ODBC manager. For PHP, include ODBC manager by linking the additional library `libodbc.so` into the PHP library. To communicate with the Informix IDS database server, at least Informix Connect is required on the machine where the Web server is running.

Steps to setup unixODBC with Informix and PHP:

1. Install ODBC manager.

- Download the ODBC manager source from:

<http://www.unixodbc.org>

- Unpack and untar the download file in the source directory.
- Configure and build the package.

```
./configure --prefix=/usr --sysconfdir=/etc --enable-gui=no
make
make install
```

2. Build PHP.

- Set the environment variable:

```
export CUSTOM_ODBC_LIBS=-L/usr/lib -lodbc
```

- Configure PHP with unixODBX and Apache:

- For Apache V2.0

```
./configure --with-unixODBC=/usr
--with-apxs2=/usr/local/apache2/bin/apxs --with-config-file-path=/etc
```

- For Apache V1.3

```
./configure --with-unixODBC=/usr
--with-apxs=/usr/local/apache1/bin/apxs --with-config-file-path=/etc
```

3. Build and install the PHP.

```
make
make install
```

- Verify if `libodbc.so` is included:

```
ldd /usr/local/bin/php
libcrypt.so.1 => /lib/libcrypt.so.1 (0x00697000)
librt.so.1 => /lib/tls/librt.so.1 (0x0037b000)
libresolv.so.2 => /lib/libresolv.so.2 (0x009a2000)
libm.so.6 => /lib/tls/libm.so.6 (0x0085a000)
libnsl.so.1 => /lib/libnsl.so.1 (0x00c42000)
```

```
libz.so.1 => /usr/lib/libz.so.1 (0x00990000)
libodbc.so.1 => /usr/lib/libodbc.so.1 (0x00727000)
libdl.so.2 => /lib/libdl.so.2 (0x00854000)
libxml2.so.2 => /usr/lib/libxml2.so.2 (0x00168000)
libpthread.so.0 => /lib/tls/libpthread.so.0 (0x0097c000)
libc.so.6 => /lib/tls/libc.so.6 (0x0038f000)
/lib/ld-linux.so.2 (0x00710000)
```

#### 4. Configure ODBC manager.

We configured the ODBC manager to look for the ini files in /etc. Two ini files can be included in /etc. odbcinst.ini is required:

- **odbc.ini**: Example 5-16 shows the content of odbc.ini.

*Example 5-16 /etc/odbc.ini*

---

```
[stores_demo]
Driver=Informix
Description=Test
Database=stores_demo
EnableInsertCursors=0
GetDBListFromInformix=1
HostName=polonium
LogonID=informix
Password=123456
Protocol=onsoctcp
ServerName=on1000UC3soc
Service=22222
DriverODBCVer=03.51
TrimBlankFromIndexName=1
```

---

- **odbcinst.ini**: Example 5-17 shows the entry:

*Example 5-17 odbcinst.ini*

---

```
[Informix]
Description=Informix
Driver=/home/informix/sdk/lib/cli/libifcli.so
APILevel=1
DriverODBCVer=03.51
FileUsage=1
SQLLevel=1
smProcessPerConnect=Y
```

---

The PEAR and PHP libraries and executable are installed in the same directories as described for Informix PDO and ifx\_.\*.



## Configuring Apache with PHP

We have Apache and PHP installed on the Web server machine. This does not mean that they are able to work together without additional configuration. There are some small changes required.

### *php.ini*

*php.ini* is the PHP configuration file. The location of this file was configured by the parameter `--with-config-file-path=<dir>`, for example, `--with-config-file-path=/etc`. Today there are products distributing their own *php.ini*. We recommend backing up the *php.ini* file after each change to preserve the latest settings and to avoid overwrites when installing a new product. In the *php.ini* file, each function set should have its own section to contain the function settings. The settings of the PHP can be checked with the `phpinfo()`.

We have covered how to compile static or shared extensions into the PHP. Shared extensions give the flexibility to add only the functions you need to the PHP. This is also the common way the Linux distributions provide the extensions. It gives a flexible way to add or remove packages on demand. Example 5-18 shows the necessary changes in the *php.ini* file for using the shared extension *odbc.so*. To use other extensions, replace the *odbc.so* with the name of the extension and make sure that the library resides in the extension directory.

#### *Example 5-18 php.ini changes for a shared unixODBC extension*

---

```
php.ini
extension_dir = /usr/lib/php4
extension = odbc.so
```

```
ls -al /usr/lib/php4
total 172
drwxr-xr-x  2 root root  4096 Jan 31 16:19 .
drwxr-xr-x 92 root root 69632 Jan 31 09:57 ..
-rwxr-xr-x  1 root root 56412 Apr 26  2005 odbc.so
```

---

You may also want to adjust two parameters *display\_errors* and *error\_reporting* in the PHP section of the *php.ini* file. The initial setting of *display\_errors* is Off and the *error\_reporting* is commonly commented out. For installation verification and a development environment, we recommend set these two parameters as:

```
display_errors = On
error_reporting = E_ALL
```

The possible settings for *error\_reporting* and their meanings are described in a comment above the parameter in the *php.ini* file. In a production environment,

the requirements for error reporting are certainly different. The application should prevent the errors shown in the browser. We strongly recommend set `display_errors` to Off.

### ***httpd.conf***

To connect the Apache with the PHP, add the following lines into the `httpd.conf` file:

For Apache V2.0:

```
LoadModule php5_module      modules/libphp5.so
AddType application/x-httpd-php .php .html
AddType application/x-httpd-php-source .phps
```

For Apache V1.3:

```
LoadModule php5_module      libexec/libphp5.so
AddType application/x-httpd-php .php .html
AddType application/x-httpd-php-source .phps
```

## **Startup and verification**

A simple way to verify the setup of Apache Web server is to start the server.

### ***Environment variables***

Make sure that the following environment variables are set before starting up the Apache Web server. Take the settings as an example.

```
export INFORMIXDIR=/home/informix/sdk
export INFORMIXSERVER=on1000UC3soc
export
LD_LIBRARY_PATH=/home/informix/sdk/lib:/home/informix/sdk/lib/cli:/home/informix/sdk/lib/esql
```

To make the environment variable settings persistent for all sessions in the system using bash, add them to `/etc/bashrc` for Red Hat, or `/etc/bash.bashrc` for SUSE. To make them only available for Apache, use the `envvars` file provided with Apache V2. It can be found in the `bin` directory of the Apache installation. This file will be read by the `apachectl` script before starting the Web server.

### ***startup***

To start the Apache Web server:

- ▶ Apache V2.0:
  - For RHEL and your own builds, use:  
`apachectl -k start`
  - For SLES9, use:

```
apache2ctl -k start
```

► Apache V1.3

```
apachectl start
```

**Note:** On SLES9, depending on the settings in the httpd.conf file, the Apache Web server may fail to start with a missing file error. The httpd.conf file of this distribution has several Include directives. One of them includes the file /etc/apache2/sysconfig.d/include.conf which does not exist by default. Either touch the file or comment out the Include in the httpd.conf file to start apache2ctl successfully.

### 5.1.6 Installation verification

Once all the products were installed and configured, the last step was to verify if all the components can work together.

#### Web server

Web server verification can be done using a Web browser. Type the machine name as the URL, and the welcome window of the Web server should appear. To verify the Web server version, click the Online-Documentation link.

On SLES9, if you have not installed the example pages rpm, you will see an access permission error message. This message can be ignored.

#### PHP with Informix interfaces

PHP with Informix interface setup verification falls into two areas:

► Connectivity with Web server

The first verification of the PHP environment is the PHP with the Web server. Use the PHP script in Example 5-11 on page 205 to verify the PHP environment setup. Run the script. If you see the HTML table with the PHP environment in the Web browser, the setup of the link (the changes in the httpd.conf file) between the PHP and the Web server was successful. You should also check the configuration command at the top of the table. It should match your settings.

► Connectivity with Informix IDS database

The next step is to verify the connectivity between the PHP and Informix IDS database server. For each interface, we provide a small PHP script which will connect to the Informix databases and try to read some rows. If the interface is configured properly, the rows will be returned and the result displayed in the browser.

## Informix PDO

Verifying PHP with Informix PDO setup:

- Connectivity with Web server

Verify the PHP with Informix PDO setting by checking the `phpinfo()` output. Make sure that the Informix is listed in the PDO table section, and that there is another table section for Informix PDO, which usually shows only the table header. See Figure 5-3.

PDO	
PDO support	enabled
PDO drivers	sqlite2, sqlite, informix
pdo_informix	
pdo_informix support	enabled

Figure 5-3 Informix PDO with `phpinfo()`

- Connectivity with Informix IDS database

Verify the database connection is done by using the script shown in Example 5-19. This script uses one of the system databases so you can use this script directly without creating a database. The browser should show a result set of five rows.

Example 5-19 Connectivity verification with Informix PDO

```
<?php
try {
    $dbh = new PDO("informix:: database=sysmaster; server=on1000UC3soc;",
        "informix", "123456");

    printf("<pre>");
    printf(" %s \n", $dbh->getAttribute(PDO::ATTR_DRIVER_NAME) );

    $stmt=$dbh->query('SELECT * FROM systables');
    $row=$stmt->fetch(PDO::FETCH_ASSOC);
    $count=0;
    while ( $stmt && $count<5 && $row)
    {
        print_r($row);
        $row=$stmt->fetch(PDO::FETCH_ASSOC);
        $count++;
    }
}
```

```

        printf("</pre>");
    } catch (PDOException $e) {
        print "Error!: " . $e->getMessage();
        exit;
    }
?>

```

---

## ***Troubleshooting***

A couple of commonly seen errors in setting the database connectivity are described here.

### ► Missing environment variable setting

Setting the environment variable INFORMIXDIR is required for starting the Apache. If this variable is not set properly, the script testing the database connectivity will also fail. Example 5-20 shows the error message. We used here the PHP command line utility to generate the output. Similar messages would occur in the browser.

#### *Example 5-20 INFORMIXDIR environment variable not set*

```

/usr/local/bin/php pdoconnect.php
Error!: SQLSTATE=HY000, SQLDriverConnect: -23101
[Informix][Informix ODBC Driver][Informix]Unspecified System Error = -23101.

```

---

### ► Mismatched settings

Another important item affecting the connectivity is the setup of the Informix run-time environment. Informix Connect or Informix SDK provides the connectivity at run time. If the settings between the environment variables and the sqlhosts do not match, you will see messages similar to the one shown in Example 5-21.

#### *Example 5-21 Database server name in connection string is wrong*

```

/usr/local/bin/php pdoconnect.php
Error!: SQLSTATE=HY000, SQLDriverConnect: -25555 [Informix]
[Informix ODBC Driver][Informix]Server on1000UC3soc1 is not listed as a
dbserver name in sqlhosts.

```

---

## ***ifx-\* function set***

Verifying PHP with ifx-\* set setup:

### ► Connectivity with Web server

Check the phpinfo() output:

- Right configuration line on the top of the HTML table

- If you have built the interface static into the PHP library, the informix table should be in the HTML output as shown in Figure 5-4. The variables in the table are either taken from the `/etc/php.ini` file or are the default settings.

informix		
Informix support		enabled
Active Persistent links		0
Active links		0
ESQL/C Version		9.60
Directive	Local Value	Master Value
<code>ifx.allow_persistent</code>	On	On
<code>ifx.blobinfile</code>	1	1
<code>ifx.byteasvarchar</code>	0	0
<code>ifx.charasvarchar</code>	0	0
<code>ifx.default_host</code>	<i>no value</i>	<i>no value</i>
<code>ifx.default_password</code>	<i>no value</i>	<i>no value</i>
<code>ifx.default_user</code>	<i>no value</i>	<i>no value</i>
<code>ifx.max_links</code>	Unlimited	Unlimited
<code>ifx.max_persistent</code>	Unlimited	Unlimited
<code>ifx.nullformat</code>	0	0
<code>ifx.textasvarchar</code>	0	0

Figure 5-4 *ifx* interface settings with `phpinfo()`

#### ► Connectivity with Informix IDS database

Example 5-22 shows the PHP script for verifying the connectivity of PHP with Informix database using `ifx_*` function set. The script selects data from a system table in a database automatically created in Informix IDS. If the database connection is successful, the result set will be returned as a simple HTML table.

Example 5-22 *First attempt to connect to the Informix server with `ifx_connect`*

```
<?php

$link = ifx_connect("sysmaster@on1000UC3soc","informix","123456");
if ( !$link ) {
    printf("</pre>");
    printf("Connect: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
    printf("</pre>");
    exit();
}

$stab_list = ifx_prepare ("SELECT tablename FROM systables",$link);
$return=ifx_do($stab_list);
```

```

if ( $return == 0 ) {
    printf("</pre>");
    printf("Select: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
    printf("</pre>");
    exit();
}

ifx_htmltbl_result($tab_list,"border=\"2\"");
ifx_free_result($tab_list);

@ifx_close($link);
?>

```

---

### ***Troubleshooting***

The ifx\_\* function set also relies on the correct settings in environment variables and the configuration file for Informix SDK, so there are several sources for problems in the first attempt to connect to the Informix IDS database server. We list some of the most common mistakes causing connectivity problems. You may notice that the error message shows twice in the output of our examples. The first message comes from the connect function for which we did not disable the error display. The second one is from the error caption. For a detailed discussion about error management, refer to Section 5.2.7, “Error handling” on page 315.

Environment variables for Informix products are necessary to enable the client to find the database server. Example 5-23 and Example 5-24 show the error messages when the environment for the client is set improperly. Here we used the PHP command line utility to generate the output. The browser would generate similar results.

#### ***Example 5-23 Missing INFORMIXDIR in the Apache environment***

---

```

/usr/local/bin/php ifxconnect.php
Warning: ifx_connect(): E [SQLSTATE=IX 001  SQLCODE=-1829] in
/usr/local/apache2/htdocs/ifxconnect.php on line 3

```

---

#### ***Example 5-24 Missing INFORMIXSERVER in the Apache environment***

---

```

/usr/local/bin/php ifxconnect.php
Warning: ifx_connect(): E [SQLSTATE=IX 000  SQLCODE=-25560] in
/usr/local/apache2/htdocs/ifx/connect/ifxconnect.php on line 3
<pre>Connect: Returncode E [SQLSTATE=IX 000  SQLCODE=-25560] -- Environment
variable INFORMIXSERVER must be set.
</pre>

```

---

After setting the right environment variables, we have to take into account that we are working in a client server architecture. So the user running the PHP script has to be authenticated at the database server machine for using the server. In Example 5-25, the connection is attempting to connect with the wrong password, which will certainly generate an error message.

*Example 5-25 User connection password is wrong*

---

```
/usr/local/bin/php ifxconnect.php
Warning: ifx_connect(): E [SQLSTATE=IX 000  SQLCODE=-952] in
/usr/local/apache2/htdocs/ifx/connect/ifxconnect.php on line 3
<pre>Connect: Returncode E [SQLSTATE=IX 000  SQLCODE=-952] -- User
(informix@lead.itsosj.sanjose.ibm.com)'s password is not correct for the
database server.
</pre>
```

---

Verify the following if all necessary Informix environment variables set in the Apache environment are verified with `phpinfo()` and you still receive an error message as shown below:

```
/usr/local/bin/php ifxconnect.php
/usr/local/bin/php: error while loading shared libraries: libifsql.so:
cannot open shared object file: No such file or directory
```

1. Does the library exist? If not, check the installation.
2. Does the user `informix` has the authority to run `/usr/local/bin/php <connectionsript>?` If not, check permissions in the installation directory.
3. Are there other users getting similar errors such as this?

In this case, check the permissions of the installation path. One of the directories above the SDK installation directory may not have read permissions for public. A typical scenario is the installation in the home directory of the `informix` user in Red Hat. The home directories have `rwX-----` as default permissions.

### ***The PEAR DB package***

Verifying PHP with PEAR DB setup:

- Connectivity with Web server

Since the DB package requires the `ifx_*` function set included in the PHP, the output for `phpinfo()` should be similar to the output from `ifx_*`.

- Connectivity with Informix IDS database

The connectivity with the Informix database for DB package can be tested using the script shown in Example 5-26 on page 223. We do not recommend using the script for the `ifx_*` verification even though it will work. From the application point of view, the function set for the PEAR DB is different. We



used here another script written with functions provided by PEAR DB. The include path where the PEAR extensions were installed is the path of our project environment. You need to change the include path to the directory where your DB.php file resides.

*Example 5-26 Connectivity verification with DB package*

---

```
<?php

ini_set("include_path","usr/local/lib/php");
require_once("DB.php");

$dsn = array(
    'phptype' => 'ifx',
    'username' => 'informix',
    'password' => '123456',
    'database' => 'sysmaster',
    'host' => 'on1000UC3soc'
);

$dbh = DB::connect($dsn);

if (DB::isError($dbh))
{
    printf("<pre> Connect failed with : %s\n</pre>", $dbh->getUserInfo());
    exit();
}

$dbh->setFetchMode(DB_FETCHMODE_ASSOC);

$sql="SELECT tabname,tabid FROM systables";
$erg=$dbh->query($sql);
$row=$erg->fetchRow();
$count=0;

printf("<pre>");
while ($row && $count <5)
{
    print_r($row);
    $row=$erg->fetchRow();
    $count++;
}
printf("</pre>");
if (DB::isError($dbh))
{
    printf("<pre> Connect failed with : %s\n</pre>", $dbh->getUserInfo());
    exit();
}
```

```
}  
?>
```

The script in Example 5-26 on page 223 will try to connect to the Informix IDS. It will open the sysmaster database, then run a SELECT querying table names from a system table and fetch the first five rows from the result set.

### ***The unixODBC interface***

Verifying PHP with unixODBC setup:

► Connectivity with Web server

Check the `phpinfo()` output to verify the PHP configuration settings. Make sure the unixODBC is in the configuration command. See Figure 5-5.

Build Date	Jan 31 2006 10:53:03
Configure Command	'./configure' '--with-unixODBC=/usr' '--with-apxs2=/usr/local/apache2/bin/apxs'

*Figure 5-5 Configure command for unixODBC*

If you compiled the unixODBC static into the PHP library, the HTML table should have an ODBC section. The default output is shown in Figure 5-6 on page 225.

odbc		
ODBC Support		enabled
Active Persistent Links	0	
Active Links	0	
ODBC library	unixODBC	
ODBC_INCLUDE	-I/usr/include	
ODBC_LFLAGS	-L/usr/lib	
ODBC_LIBS	-lodbc	
Directive	Local Value	Master Value
odbc.allow_persistent	On	On
odbc.check_persistent	On	On
odbc.default_db	no value	no value
odbc.default_pw	no value	no value
odbc.default_user	no value	no value
odbc.defaultbinmode	return as is	return as is
odbc.defaultlrl	return up to 4096 bytes	return up to 4096 bytes
odbc.max_links	Unlimited	Unlimited
odbc.max_persistent	Unlimited	Unlimited

Figure 5-6 unixODBC PHP parameters

Another way to check if the manually created PHP has successfully included the unixODBC is using the php command:

```
nm /usr/local/bin/php | grep odbc_conn
080d1bd8 t _close_odbc_conn
080d5a30 T zif_odbc_connect
```

#### ► Connectivity with Informix IDS database

The connection to the Informix database can be checked with PHP script shown in Example 5-27. It will return the first two tables from the sysmaster database.

Example 5-27 Verify the database connectivity for unixODBC interface

```
<?php
$cn = odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=sysmaster",
"informix","123456");
if ( !$cn ) {
    printf("<pre>Connect: attempt to connect was not successful \n</pre>");
    exit();
}
$str="SELECT * FROM systables";
```

```

$odbc_result = odbc_exec($cn,$str);
if ( !$odbc_result ) {
    printf(" %s %s \n",odbc_error($cn), odbc_errormsg($cn));
    exit();
}

printf("<pre>");
odbc_fetch_row($odbc_result);
printf(" %s %s %s \n",odbc_result($odbc_result,1), odbc_result($odbc_result,2),
odbc_result($odbc_result,3));

odbc_fetch_row($odbc_result);
printf(" %s %s %s \n",odbc_result($odbc_result,1), odbc_result($odbc_result,2),
odbc_result($odbc_result,3));

printf("</pre>");
@odbc_close($cn);
?>

```

---

### ***Troubleshooting***

In the case of unixODBC, we use a ODBC manager for maintaining the connectivity. Your focus for troubleshooting, in view of configuration files, for finding the source of the problem is quite different.

As discussed in “The unixODBC interface” on page 213, there are two files defining the connection to the data source. If you miss creating odbcinst.ini, you will get an error as shown in Example 5-28.

#### ***Example 5-28 odbcinst.ini file missing***

---

```

/usr/local/bin/php odbconnect.php
Warning: odbc_connect(): SQL error: [unixODBC] [Driver Manager]Data source name
not found, and no default driver specified, SQL state IM002 in SQLConnect in
/usr/local/apache2/htdocs/odbconnect.php on line 4

```

---

Another common source for connection errors is again a missed environment variable setting. odbcinst.ini points to the complete path for the Informix connectivity library. There are other shared libraries needed by this library which do not reside in the same directory. Missing the directories in the LD\_LIBRARY\_PATH will generate the output as shown in Example 5-29.

#### ***Example 5-29 LD\_LIBRARY\_PATH missing in the Apache environment***

---

```

/usr/local/bin/php odbconnect.php

Warning: odbc_connect(): SQL error: [unixODBC] [Driver Manager]Can't open lib
'/home/informix/sdk/lib/cli/libifcli.so' : libifgls.so: cannot open shared

```

---

## 5.2 Application development with Informix IDS

Maintaining the data within a database server is only one of the key tasks in the modern Web application environment. Secured data store and fast and reliable access are the common requirements. These requirements lead to the best efforts in implementing the database interface into the application.

The following sections provide an in-depth description about how to use the specific features of Informix IDS interfaces. We show the capabilities of the interfaces by easy-to-rewrite examples. We cover the following major database and database application development concepts:

- ▶ How to set up connections to Informix IDS.
- ▶ Dynamic versus static INSERT, UPDATE, and DELETE statements.
- ▶ Using SELECT statements, cursors, and result sets in PHP.
- ▶ Using stored procedures on the database server, error handling, and tracing.
- ▶ Using SQL statements and PHP extension functions for transaction management.
- ▶ Using complex data types such as row types and collections.
- ▶ Large data types like BLOBs and SBLOBs (Smart BLOBs).
- ▶ Errors and exceptions in PHP.
- ▶ Basic application and server optimization.
- ▶ Using SQL additions of Informix XPS database server in PHP.

In a heterogeneous database environment, keeping the code portable is a basic requirement for the application. The application should be able to run against different databases with minimum or no changes. Since the PDO interface is designed to support this, we focus more on the Informix PDO examples.

The Car Portal application data model introduced in Chapter 2, “Sample scenario description” on page 31 is used for examples discussed here.

### 5.2.1 Connection to the database

From the initial Car Portal Web page, a user can use the application as a car dealer or as a customer. Once the customer or the dealer has completed filling

out the registration or login form in the browser, a connection to the database has to be established.

Regardless of which PHP extension is used, you need to decide what connection type to use for database connection and what user type to use for authentication.

► Connection type

The connection type behavior is controlled by the Web server. There are two connection types. For the persistent connection, the Web server will leave the connection to the database open after the PHP script finishes its work. The next attempt to connect to the database with the same parameters will reuse the connection. For the standard connection, the database connection will be closed when the script finishes its execution.

Here are some considerations for persistent connection:

- When using the persistent connection, the same connection will be used for the next request with the same connection parameters. This could cause some kind of connection pooling depending on the implementation and how you decide if the connection will be reused or not. Unintentional connection pooling is most likely a source of serious problems, even if you use transactions.
- Opening up a persistent connection only makes sense in a Web server environment if the connection will not be closed in the same script with a connection close call. Calling the PHP command line processor on the shell will close the connection to Informix IDS at the end of the execution of the script.
- The Web server spawns more than one process to handle the incoming Web requests, so reusing the existing persistent connection is only valid for a specific Web server process. So it easily results in the database server having more connections open than actual running Web server processes. Example 5-30 shows an onstat output which may illustrate this behavior.
- Restarting the database server without cleaning up the remaining persistent connections in the Web server environment will produce no errors at connection time when the connection is reused, but will fail at the first database statement. This type of error is hard to diagnose.

*Example 5-30 Multiple sessions open even with persistent connections*

---

```
informix@polonium:~/v10> onstat -g sql
```

```
IBM Informix Dynamic Server Version 10.00.UC3R1  -- On-Line -- Up 00:10:18 --  
86636 Kbytes
```

```
Sess  SQL              Current          Iso Lock      SQL  ISAM F.E.
```

Id	Stmt type	Database	Lvl	Mode	ERR	ERR	Vers	Explain
39	-	vps	CR	Not Wait	0	0	9.03	Off
29	-	vps	CR	Not Wait	0	0	9.03	Off
24	-	vps	CR	Not Wait	0	0	9.03	Off

---

#### ► The user type

The user type is controlled by the database and has to do with authentication at the database server. You can choose an authentication with user ID and password in the connection string, or a trusted user connect where the database server does not apply an authentication. This will be controlled by settings on operating system resources, such as `.rhosts` or `/etc/hosts.equiv` files.

At the end of our introduction into the database connections, we discuss some remarks related to security and connections. Since you have to specify user names and passwords in the PHP scripts, pay higher attention to these files in regard to security. Use encapsulation for files containing the passwords and put these files in directories secured with the `.htaccess` or `httpd.conf` mechanism of the Apache Web server.

## Connections with Informix PDO

Informix PDO requires, at connection time, a defined set of parameters: database name and database server name. We prefer you require a user ID and password for untrusted user sessions. There are some optional parameters set which can influence the cursor behavior. In this section, we focus on the required connection parameters. For further description of the optional parameters, refer to “SELECT statements with Informix PDO” on page 248.

Example 5-31 gives you a detailed overview about the various parameter settings in an attempt to connect to the Informix IDS. Choose one of the connection statements which meets your requirements.

### *Example 5-31 Simple database connection statements in Informix PDO*

---

```
/* standard connect */
$dbh = new PDO("informix:; database=sysmaster; server=on1000UC3soc;",
"informix", "123456");

/* standard connect trusted user */
$dbh = new PDO("informix:; database=sysmaster; server=on1000UC3soc;");

/* persistent connect untrusted user */
$dbh = new PDO("informix:; database=sysmaster; server=on1000UC3soc;"
,"informix","123456",array(PDO::ATTR_PERSISTENT=> true));

/* persistent connect trusted user */
```

```
$dbh = new PDO("informix:: database=sysmaster; server=on1000UC3soc;"
,NULL,NULL,array(PDO::ATTR_PERSISTENT=> true));
```

---

## Connections with ifx\_\*

The `ifx_connect` and `ifx_pconnect` are the functions you have to use for connection to the database in this interface. For untrusted sessions, which we prefer, the database, user ID, and password parameters have to be specified. For trusted sessions, the database is sufficient to get a successful connection. Example 5-32 shows a detail of available connection calls.

*Example 5-32 Simple database connection requests in ifx\_\* functions*

---

```
/* standard untrusted */
$link = ifx_connect("sysmaster@on1000UC3soc","informix","123456");

/* persistent untrusted*/
$link = ifx_pconnect("sysmaster@on1000UC3soc","informix","123456");

/* standard trusted */
$link = ifx_connect("sysmaster@on1000UC3soc");

/* persistent trusted */
$link = ifx_pconnect("sysmaster@on1000UC3soc");
```

---

## Connections with PEAR DB

PEAR DB is more restrictive when specifying the connecting user. We have not found a way to specify a connection string without user and password. Example 5-33 gives you an overview about the possible formats of a connection string for a standard connection.

*Example 5-33 Simple database connections with PEAR DB*

---

```
$dsn = array(
    'phptype' => 'ifx',
    'username' => 'informix',
    'password' => '123456',
    'database' => 'vps',
    'host' => 'polonium'
);

$dbh = DB::connect($dsn);

/* or */
/* User informix */
/* Password 123456 */
/* communication type tcp */
/* database server name on1000UC3soc */
```



```

/* database port 1526 */
/* database name vps */

$dbh = DB::connect("ifx://informix:123456@tcp+on1000UC3soc:1526/vps");
$dbh = DB::connect("ifx://informix:123456@on1000UC3soc:1526/vps");
$dbh = DB::connect("ifx://informix:123456@on1000UC3soc/vps");

```

To use a persistent connection, you need to add an additional options array parameter to the connect call, see Example 5-34 for a piece of code to use.

*Example 5-34 Persistent database connections with PEAR DB*

```

$dsn = array(
    'phptype' => 'ifx',
    'username' => 'informix',
    'password' => '123456',
    'database' => 'stores_demo',
    'host' => 'polonium'
);

$options = array(
    'persistent' => True
);
$dbh = DB::connect($dsn,$options);

```

## Connections with unixODBC

For a connection with unixODBC, similar parameters are required as for Informix PDO. You have to specify the server name, the database name, and the section header name of your connectivity specifications for the ODBC manager in the `odbcinst.ini` file. User name and password are required in the case of an untrusted connection request. There are additional optional parameters controlling the cursor behavior which are discussed in detail in “Selects with unixODBC” on page 265. Example 5-35 shows the different parameter settings for the connection to an Informix IDS database server.

For the unixODBC, the parameters in the connection string for a Windows system are different from those for a Linux system. Refer to the script in Example 5-112 on page 351 for the Windows example.

```

$cn = odbc_connect("DSN=unixODBC1;", "informix", "123456")

```

*Example 5-35 Simple database connections with unixODBC*

```

/*trusted*/
$cn = odbc_connect
("Driver=Informix;Server=on1000UC3soc;Database=sysmaster",NULL,NULL);

```

```

/* not trusted */
$cn = odbc_connect
("Driver=Informix;Server=on1000UC3soc;Database=sysmaster","informix","123456");

/* persistant not trusted*/
$cn = odbc_pconnect
("Driver=Informix;Server=on1000UC3soc;Database=sysmaster","informix","123456");

/* persistant trusted*/
$cn = odbc_pconnect
("Driver=Informix;Server=on1000UC3soc;Database=sysmaster",NULL,NULL);

```

---

## 5.2.2 Static and dynamic SQL statements in PHP

Informix IDS supports both static and dynamic SQL statements for the client applications, such as PHP programs. Dynamic SQL in difference to static SQL provides the ability to work with a placeholder such as “?” on predefined places in the statement. This placeholder will be replaced with the real value at execution time. The advantage of using dynamic SQL statements is that the statement will be parsed and optimized once, but can be executed multiple times with variable statement parts. Common examples for dynamic SQL are inserts or select, update, or delete statements with variable filter conditions.

When working with the placeholders, you have to decide to provide the replacement either on the database server side (in Informix PDO and unixODBC), or provide the placeholder’s replacement in the PHP extension itself (PEAR DB). We provide an in-depth discussion about the abilities of handling dynamic and static SQL of several PHP extensions with a major focus on the insert statement, but we also give some short examples for DELETE and UPDATE statements.

### Informix PDO and static SQL statements

In the car dealer application, the cars for sale presented in the Web site are from the CAR table. Example 5-36 shows various ways of executing a static INSERT with Informix PDO. The script inserts data to the CAR table. At the end of the script, you can see an additional example of how to determine the number of rows affected by the INSERT statement and how to figure out the serial number for the last INSERT statement.

*Example 5-36 Possible ways to execute INSERT statements with Informix PDO*

---

```

<?php

$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");

```

```

if (!dbh) {
    exit();
}
/*
    Using PDO::exec for the insert
*/

$statement="INSERT INTO car VALUES
(0,'SUV','Temia','1971','Black','Towing','N','Y','N','6',97077,'Sold
as-is.');"

$dbh->exec($statement);
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" execute insert failed with %s \n",$error["1"]);
    exit(1);
}
/*
    Using PDO::query for the insert
*/

$dbh->query($statement);
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" execute insert failed with %s \n",$error["1"]);
    exit(1);
}

/*
    Using PDO::prepare
    and PDO::Statement::execute for the insert
*/
$stmt=$dbh->prepare($statement);
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" prepare insert failed with %s \n",$error["1"]);
    exit(1);
}
$stmt->execute();
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" execute insert failed with %s \n",$error["1"]);
    exit(1);
}

printf("RowCount: %d \n",$stmt->rowCount());

$stmt=$dbh->query("SELECT MAX(t_id) FROM car");
$row=$stmt->fetch(PDO::FETCH_NUM);

```

```
printf("LastInserted %d \n",$row[0]);
?>
```

---

In a Web application, most of the time the input data comes from a HTML form entered by a Web user. Example 5-37 shows, based on a very simple HTML form, how the data flows from the HTML over the PHP script into the database table. The array \$\_POST contains all the settings for the buttons and the values for the text fields from the HTML.

*Example 5-37 HTML and PHP for inserting format values with static SQL in PDO*

---

#### **basic html format**

```
<?php
printf("<form method=\"post\" action=\"simplefromhtml.php\" >");
printf("type <input name=\"type\" /><br />");
printf("model <input name=\"model\" /><br />");
printf("year <input name=\"year\" /><br />");
printf("color <input name=\"color\" /><br />");
printf("package <input name=\"package\" /><br />");
printf("AC <input name=\"ac\" /><br />");
printf("Windows <input name=\"powerwindows\" /><br />");
printf("Automatic <input name=\"automatic\" /><br />");
printf("Doors <input name=\"doors\" /><br />");
printf("Price <input name=\"price\" /><br />");
printf("Extras <input name=\"extra\" /><br />");
printf("<input type=\"submit\" value=\"Confirm\" name=\"Button\"/>");
printf("<input type=\"submit\" value=\"Abort\" name=\"Cancel\"/>");
printf("</form>");
?>
```

#### **PHP Script handling the insert**

```
<?php
$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");
if (!$dbh) {
    exit();
}

/*
data from the Input Format
0|SUV|Temia|1971|Black|Towing|N|Y|N|6|97077|Sold as-is.
*/

$statement="INSERT INTO car VALUES (0,".
"" . $_POST["type"] . "',' .
"" . $_POST["model"] . "',' .
```

```

"" . $_POST["year"] . "','" .
"" . $_POST["color"] . "','" .
"" . $_POST["package"] . "','" .
"" . $_POST["ac"] . "','" .
"" . $_POST["powerwindows"] . "','" .
"" . $_POST["automatic"] . "','" .
"" . $_POST["doors"] . "','" .
$_POST["price"] .
"', '" . $_POST["extra"] . "');"

$dbh->query($statement);
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" prepare insert failed with %s \n",$error["1"]);
exit(1);
}
?>

```

---

Finally, we introduce a housekeeping PHP script in our sample application using static SQL for UPDATE and DELETE statements. Example 5-38 shows how to set the promotion end date to a month later and how to delete old reviews from the database.

---

*Example 5-38 Static updates and deletes with Informix PDO*

---

```

<?php
$dbh = new PDO("informix:host=polonium; service=22222; database=vps1;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");
if (!$dbh) {
    exit();
}
/*
static update advance all promotion + one month
*/

$statement="UPDATE promotion SET p_end_date=p_end_date+30 ";

$stmt=$dbh->query($statement);
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" update failed with %s \n",$error["1"]);
}
printf("#Promotions updated: %d \n",$stmt->rowCount());
/*
delete all reviews older than year
*/
$dbh->beginTransaction();

```

```

$statement="DELETE FROM review WHERE r_date< today -365 ";
$stmt=$dbh->prepare($statement);
$stmt->execute();
printf("#Reviews deleted: %d \n",$stmt->rowCount());
if ($stmt->rowCount())>10000
{
$dbh->rollback();
}
else {
$dbh->commit();
}
?>

```

---

## IFX\_\* and static SQL statements

Example 5-39 shows the functions provided by ifx\_\* to execute an INSERT statement. The table used in this example is the CUSTOMER table in the Dealership application. Registration is required for a new customer.

At the end of the example, we also show how to get information from SQL Communication Area (SQLCA). The program gets the last inserted serial value and the amount of rows executed by this statement using the ifx\_getsqlca() function. This is similar to the use of the SQLCA structure in native ESQL/C programming. For a complete description of the values in the SQLERRD structure, refer to *ESQL/C Programmer's Manual, Version 9.53*, G251-1342.

*Example 5-39 How to execute static inserts in ifx\_\**

---

```

<?php

$link = ifx_connect("vps@on1000UC3soc","informix","123456");
if ( !$link ) {
    printf("<pre>");
    printf("Connect: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
    printf("</pre>");
    exit();
}

$statement="INSERT INTO customer VALUES
(0,'Mrs.','Angela','Clinny','Queen','Ottawa','Saskatchewan','CODE','8337808','4
545622',2329128,'97650',NULL);"
/*
    Using ifx_prepare/ifx_do for the insert
*/

$stmt=ifx_prepare($statement,$link);
ifx_do($stmt);

```

```

/*
Affected rows ?
Inserted Serial Value ?
*/

$sqlca = ifx_getsqlca ($stmt);
printf(" Serial Value returned : %d \n",$sqlca["sqlerrd1"]);
printf(" # Rows inserted : %d %d ",$sqlca["sqlerrd2"],ifx_affected_rows
($stmt));

/*
        Using ifx_query for the insert
*/

ifx_query($statement,$link);

@ifx_close($link);
?>

```

---

## PEAR DB and Static SQL statements

Example 5-40 shows how to do the INSERT statement using the PEAR DB interface. In this example, we use the PROMOTION table in our Dealership application. The PROMOTION table is another frequently used table which holds the car promotion program.

There is one important point we have to bring to your attention in this example. To use the PEAR DB, which is a set of PHP scripts, you have to add the include path for the root PHP script DB.php. This can be done by a call to the ini\_set() function.

### *Example 5-40 Static SQL within PEAR DB*

---

```

<?php

ini_set("include_path","usr/local/lib/php");
require_once("DB.php");

$dbh = DB::connect("ifx://informix:123456@on1000UC3soc/vps");

/*
        Using DB::prepare and DB::execute
*/

$stmt = $dbh->prepare('INSERT INTO promotion VALUES (0,1000,0.3,
today,today+30);');
if (PEAR::isError($stmt)) {

```

```

        die($sth->getMessage());
    }

    $dbh->execute($stmt);
    if (PEAR::isError($stmt)) {
        die($stmt->getMessage());
    }
    printf("#rows : %d \n", $dbh->affectedRows());

    /*
        Using DB::query
    */

    $result= & $dbh->query
        ('INSERT INTO promotion VALUES (0,1000,0.3, today,today+30);');
    if (PEAR::isError($stmt)) {
        die($dbh->getMessage());
    }
    /*
    not possible insert doesnt return resultset
    printf("#rows : %d \n",$result->numRows());
    */
    printf("#rows : %d \n", $dbh->affectedRows());
?>

```

---

Using PEAR DB, you are not able to get the last value for an insert into a serial column because the result set is freed in the extension function before returning the status to the PHP script. If this value is needed, you have to advance the DB\_ifx class similar to the usage of the already existing `affectedRows()` method. Example 5-41 shows you a possible solution.

---

*Example 5-41 Changes to enable PEAR DB to return the last serial value*

---

Note: All changes are related to the file `ifx.php` provided by the PEAR DB package, we will only show the code parts for the changes, not the whole file.

```

.....
/**
 * The number of rows affected by a data manipulation query
 * @var integer
 * @access private
 */
var $affected = 0;
var $serial = 0;                                <- add here
.....

```



```

function simpleQuery($query)
{
    $ismanip = DB::isManip($query);
    $this->last_query = $query;
    $this->affected = null;
    $this->serial = null;          <-- add here
    .....

    if (!$result) {
        return $this->ifxRaiseError();
    }
    $this->affected = @ifx_affected_rows($result);

    if (preg_match('/(INSERT)/i', $query)) {    <-- add here
        $sqlca = ifx_getsqlca ($result);       <-- add here
        $this->serial = $sqlca["sqlerrd1"];     <-- add here
    }                                           <-- add here

    .....

function affectedRows()
{
    if (DB::isManip($this->last_query)) {
        return $this->affected;
    } else {
        return 0;
    }
}

function serial()                                <-- add this function
{
    if (DB::isManip($this->last_query)) {
        return $this->serial;
    } else {
        return 0;
    }
}

}

.....

```

---

## unixODBC and static SQL statements

For the discussion of static SQL in unixODBC, we use the NEWORDER table. A new order will be inserted into this table once a customer finds the desired car and submits an order. Example 5-42 shows the different ways an INSERT statement can be executed using unixODBC interface.

*Example 5-42 unixODBC and static inserts*

---

```
<?php
```

```

$cn =
odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=vps","informix","123
456"
);
if ( !$cn ) {
    printf("<pre>Connect: attempt to connect was not successful \n</pre>");
    exit();
}
/*
    Using odbc_exec for the insert
*/
$str="INSERT INTO neworder VALUES ( 10000,9600, 800, 1000, 11400 ,\"cheque
\")";

$odbc_result = odbc_exec($cn,$str);
if ( !$odbc_result ) {
    printf(" %s %s \n",odbc_error($cn), odbc_errormsg($cn));
    exit();
}

/*
    Using odbc_prepare
    and odbc_exec for the insert
*/
$stmt = odbc_prepare ($cn, $str);
odbc_execute($stmt);

printf(" #rows : %d \n",odbc_num_rows ($stmt));
@odbc_close($cn);
?>

```

---

## Dynamic SQL and Informix PDO

Informix PDO provides several ways to use dynamic SQL. You have to use placeholders, such as “?” or “:parameter”, to prepare a statement for dynamic usage. These placeholders will later be substituted with the actual values. The the prepare once execute multiple times capability of dynamic statements allows you to specify different values at every execution time. To find out which statements and clauses allow placeholders, refer to *IBM Informix Guide to SQL: Syntax*, G251-2284-02. This document also is a comprehensive reference of Informix IDS SQL statements.

In Example 5-36 on page 232, we show different static INSERT statements using Informix PDO. Example 5-43 is the dynamic SQL counterpart.

```
<?php

$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");
if (!$dbh) {
    exit();
}
/*
    dynamic inserts with "?" and bindParam
*/

$stmt=
$dbh->prepare("INSERT INTO car(t_id,t_type,t_model,t_price) VALUES
(0,?,?,?)");
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" prepare insert1 failed with %s \n",$error["1"]);
    exit(1);
}

$stmt->bindParam(1, $_POST["type"]);
$stmt->bindParam(2, $_POST["model"]);
$stmt->bindParam(3, $_POST["price"]);

$stmt->execute();
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" prepare insert2 failed with %s \n",$error["1"]);
    exit(1);
}

/*
    dynamic inserts with :<> placeholders and bindParam
*/
$stmt=
$dbh->prepare("INSERT INTO car(t_id,t_type,t_model,t_price) VALUES
(0,:p1,:p2,:p3)");

$stmt->bindParam(':p1', $_POST["type"]);
$stmt->bindParam(':p2', $_POST["model"]);
$stmt->bindParam(':p3', $_POST["price"]);

$stmt->execute();
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" prepare insert2 failed with %s \n",$error["1"]);
    exit(1);
}
```

```

}

/*
    dynamic inserts with "?" placeholders and bindValue
*/
$stmt=
$dbh->prepare("INSERT INTO car(t_id,t_type,t_model,t_price) VALUES
(99,?,?,?)");

$stmt->bindValue(1, $_POST["type"],PDO::PARAM_STR);
$stmt->bindValue(2, $_POST["model"],PDO::PARAM_STR);
$stmt->bindValue(3, $_POST["price"],PDO::PARAM_STR);

$stmt->execute();
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" prepare insert3 failed with %s \n",$error["1"]);
    exit(1);
}

/*
    dynamic inserts with :<> placeholders and bindValue
*/
$stmt=
$dbh->prepare("INSERT INTO car(t_id,t_type,t_model,t_price) VALUES
(199,:p1,:p2,:p3)");

$stmt->bindValue(':p1', $_POST["type"],PDO::PARAM_STR);
$stmt->bindValue(':p2', $_POST["model"],PDO::PARAM_STR);
$stmt->bindValue(':p3', $_POST["price"],PDO::PARAM_STR);

$stmt->execute();
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" prepare4 insert failed with %s \n",$error["1"]);
    exit(1);
}
?>

```

---

Another very useful application of an insert with dynamic SQL is the implementation of the LOAD statement outside of dbaccess. dbaccess is a utility that provides the LOAD statement as an Informix extension for SQL to load delimited data from a file into tables. For PHP, besides calling dbaccess with a system call, a similar functionality can be easily implemented. Example 5-44 on page 243 shows a PHP program which reads data from a delimited data file row by row, converts it into an array and inserts it into a target table, executing the same insert multiple times with the array as a variable parameter.

For our Dealership application, with this function, the car dealer can “load” a batch of new car information in one step, and save a lot of time compared to using the HTML form and enter the car data one by one.

*Example 5-44 Simple solution for LOAD statement in Informix PDO*

---

```
<?php

$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");
if (!$dbh) {
    exit();
}

$stmt= $dbh->prepare("INSERT INTO car_test VALUES (?,?,?,?,?,?,?,?,?,?,?)");
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" prepare insert failed with %s \n",$error["1"]);
    exit(1);
}

$handle = fopen ("CAR.UNL", "r");
if ($handle) $buffer = fgets($handle, 4096);
if ($buffer ) $arr=explode("|", $buffer);

if ($arr) {
    for ($cnt=1; $cnt< 13 ; $cnt ++ ) {
        $stmt->bindParam($cnt, $arr[$cnt-1]);
    }
}

do {
    $arr=explode("|", $buffer);
    $stmt->execute();
    $error=$dbh->errorInfo();
    if ( $error["1"]) printf(" Insert failed with %s \n",$error["1"]);
    $buffer = fgets($handle, 4096);
}
while (!feof($handle) && $buffer);
fclose ($handle);

?>
```

---

Example 5-45 on page 244 shows dynamic DELETE and UPDATE with Informix PDO. For a Web application, preventing improper language in the written comments is not possible. A clean-up routine can be useful. Example 5-45 is the

housekeeping routine for the Dealership application to clean up the reviews with improper language.

*Example 5-45 Using dynamic DELETE and UPDATE with Informix PDO*

---

**PHP script for beautification of reviews**

```
<?php

$dbh = new PDO("informix:host=polonium; service=22222; database=vps1;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");
if (!$dbh) {
    exit();
}
/*
review checker for fwords
*/

$fwords=array("junk","garbage");
$generic_text="This Car was the best car I have ever bought, Good Service, Good
quality";

$stmt="UPDATE  review SET r_data=? WHERE r_data MATCHES ? OR r_data
MATCHES ? ";
$stmt= $dbh->prepare($statement);

$stmt->bindParam(1, $generic_text);
$stmt->bindParam(2, $fwords[0]);
$stmt->bindParam(3, $fwords[1]);

$stmt->execute();
?>
```

**PHP script for delete customers identified as Review spammer**

```
<?php

$dbh = new PDO("informix:host=polonium; service=22222; database=vps1;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");
if (!$dbh) {
    exit();
}
/*
cleanup review spammer
*/

$fwords=array("junk","garbage");
```

```

$statement="DELETE FROM customer WHERE c_id IN ( SELECT f_c_id FROM review
WHERE r_data MATCHES ? OR r_data MATCHES ? )";
$stmt= $dbh->prepare($statement);
$error=$dbh->errorInfo();
$stmt->execute($fwords);
?>

```

---

## Dynamic SQL and ifx\_\*

The ifx\_\* interface does not support dynamic statements with one exception. If the statement includes a reference to a column whose data type is BLOB, this reference must use the “?” placeholder. During execution time, the parameter the statement refers to must be an array of BLOBs. This will be discussed in detail in 5.2.6, “BLOB and SBLOB data types” on page 303.

## Dynamic SQL and PEAR DB

Since ifx\_\* does not support dynamic SQL, PEAR DB has included a simulation of preparing the statement. Instead of sending the statement for prepare to the server, the statement will be transformed in the DB package. The data is passed in the subsequent call of EXECUTE. The placeholder will be replaced with the parameter data and the statement sent to the server for execution.

Another solution provided by DB is an auto insert. A similar interface also exists for an update. Example 5-46 shows the usage for both implementations. We use the script to insert promotions.

### *Example 5-46 PEAR DB dynamic inserts and auto inserts*

---

```

<?php

ini_set("include_path","usr/local/lib/php");
require_once("DB.php");

$dbh = DB::connect("ifx://informix:123456@on1000UC3soc/vps");

/*
    dynamic inserts with ? and prepare and execute
*/

$startdate=date("m/d/y");
$enddate = date("m/d/y",mktime(0, 0, 0, date("m")+1 , date("d"), date("Y")));
$data = array(0, 1050,0.3,$startdate,$enddate );

$stmt = $dbh->prepare('INSERT INTO promotion VALUES (?,?,, ?,?);');
if (DB::isError($stmt)) {
    die($dbh->getMessage());
}

```

```

$dbh->execute($stmt,$data);
if (DB::isError($stmt)) {
    die($stmt->getMessage());
}

/*
    Autoprepere
*/

$startdate=date("m/d/y");
$enddate =date("m/d/y",mktime(0, 0, 0, date("m")+1 , date("d"), date("Y")));
$columns=array("p_id","f_m_id","p_amount","p_start_date","p_end_date");
$data = array(0, 1050,0.3,$startdate,$enddate );

$stmt = $dbh->autoPrepare("promotion", $columns, DB_AUTOQUERY_INSERT);
if ($stmt)
$dbh->execute($stmt,$data);
?>

```

---

## Dynamic SQL and unixODBC

unixODBC supports dynamic SQL with limitations. Only the INSERT, DELETE, or UPDATE SQL statements without returning a result set can be dynamic. odbc\_execute executing a dynamically prepared statement is not able to return any other value than the status.

Example 5-47 shows the dynamic SQL usage using unixODBC. This example is a conversion from the static SQL example, Example 5-42 on page 239.

### *Example 5-47 Dynamic inserts in unixODBC*

---

```

<?php

$cn =
odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=vps","informix","123
456");
if ( !$cn ) {
    printf("<pre>Connect: attempt to connect was not successful \n</pre>");
    exit();
}
/*
    Using odbc_exec for the insert with '?' and parameter list
    neworder
*/
$parm1=10000;
$parm2=9600;
$parm3=800;
$parm4=1000 ;

```



```

$parm5=11400;
$parm6="cheque";

$parameterlist = array ( 10000,9600, 800, 1000, 11400 ,"cheque");

$str="INSERT INTO neworder VALUES ( ?,?, ?, ?, ? , ?)";
$stmt = odbc_prepare ($cn, $str);
if ( !$stmt ) {
    printf("PREPARE %s %s \n",odbc_error($cn), odbc_errormsg($cn));
    exit();
}

/*
    compilation of the different Variables to an array at exec time
*/

$odbc_result = odbc_execute($stmt,
    array($parm1,$parm2,$parm3,$parm4,$parm5,$parm6));
/*
same Insert with array variable
*/

$odbc_result = odbc_execute($stmt,$parameterlist);

@odbc_close($cn);
?>

```

---

### 5.2.3 Cursor and result set

Reliable and fast data retrieval from the database server is a major requirement of a Web application. Cursors and result sets are the basic mechanism of a database interface to retrieve data from the clients. There are three types of cursors: standard cursors, scroll cursors, and cursors with hold. *Standard cursors* allow the application to move row by row in a forward direction through the result set. *Scroll cursors* have the capability to navigate the result set in each direction with an offset. This includes the ability to position at the beginning and at the end of the result set. *Cursors with hold* give another option of leaving the cursor open when committing a transaction. If the application logic has committed or rolled back a transaction and required the cursor to remain opened, the cursor has to be declared as a hold cursor.

In the following sections, we provide a detailed discussion about how a SELECT statement can be included in the application using several PHP database interfaces. In the examples, we continue focusing on the Dealership dealer application and its base tables.

This section discusses the use of cursors and the result sets in PHP with various Informix IDS database extensions.

## SELECT statements with Informix PDO

In contrast to DELETE and UPDATE statements, SELECT SQL statements will return not only a status but also a result set if data exists. Informix PDO provides various functions to fetch data into a result set using different data structures, such as associative arrays, objects, or class instances. You can specify the data structure as a parameter in the fetch call. You are also able to define a global data structure for the “actual statement” with setFetchMode. The “actual statement” refers to the statement currently being executing. Example 5-48 shows how to fetch a result row and how to access the data structure afterwards.

*Example 5-48 Simple cursors in Informix PDO and how to represent the results*

---

```
<?php

try {
$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");
if (!$dbh) { exit(); }

    $stmt=$dbh->query('SELECT t_type,t_model,t_price FROM car WHERE t_id>1000');
    /*
5 rows into an assoc array
*/
    printf("<pre>\n");
    printf("CARS with assoc arrays \n\n");
    $row=$stmt->fetch(PDO::FETCH_ASSOC);
    $count=0;
    while ( $stmt && $count<5 && $row)
    {
        printf(" %s %s %s \n",$row["T_TYPE"],$row["T_MODEL"],$row["T_PRICE"]);
        $row=$stmt->fetch(PDO::FETCH_ASSOC);
        $count++;
    }
    printf("</pre>");

    /*
5 rows into a numerated Array
*/
    printf("<pre>\n");
    printf("CARS with num arrays \n\n");
    $row=$stmt->fetch(PDO::FETCH_NUM);
    $count=0;
    while ( $stmt && $count<5 && $row)
```

```

    {
        printf(" %s %s %s \n",$row[0],$row[1],$row[2]);
        $row=$stmt->fetch(PDO::FETCH_NUM);
        $count++;
    }
    printf("</pre>");

/*
5 rows into an object
*/
    printf("<pre>\n");
    printf("CARS with an Object \n\n");
    $row=$stmt->fetch(PDO::FETCH_OBJ);
    $count=0;
    while ( $stmt && $count<5 && $row)
    {
        printf(" %s %s %s \n",$row->T_TYPE,$row->T_MODEL,$row->T_PRICE);
        $row=$stmt->fetch(PDO::FETCH_OBJ);
        $count++;
    }
    printf("</pre>");

/*
5 rows into an hybrid array structure enum and assoc
*/
    printf("<pre>\n");
    printf("CARS with an assoc and num index \n\n");
    $row=$stmt->fetch(PDO::FETCH_BOTH);
    $count=0;
    while ( $stmt && $count<5 && $row)
    {
        printf(" %s %s %s \n",$row["T_TYPE"],$row[1],$row["T_PRICE"]);
        $row=$stmt->fetch(PDO::FETCH_BOTH);
        $count++;
    }
    printf("</pre>");

/*
5 rows into an hybrid array structure/object enum/assoc/object
*/
    printf("<pre>\n");
    printf("CARS with an assoc and num index \n\n");
    $row=$stmt->fetch(PDO::FETCH_LAZY);
    $count=0;
    while ( $stmt && $count<5 && $row)
    {

```

```

        printf(" %s %s %s \n",$row["T_TYPE"],$row[1],$row->T_PRICE);
        $row=$stmt->fetch(PDO::FETCH_LAZY);
        $count++;
    }
    printf("</pre>");

    $t_type="";
    $t_model="";
    $t_price="";
    $stmt->bindColumn(1, $t_type, PDO::PARAM_STR,50);
    $stmt->bindColumn(2, $t_model, PDO::PARAM_STR,50);
    $stmt->bindColumn(3, $t_price, PDO::PARAM_STR,50);

    /*
    5 rows fetch into a bound output variable
    */
    printf("<pre>\n");
    printf("CARS with  fetch into bould variables \n\n");
    $row=$stmt->fetch(PDO::FETCH_BOUND);
    $count=0;
    while ( $stmt && $count<5 && $row)
    {
        printf(" %s %s %s \n",$t_type,$t_model,$t_price);
        $row=$stmt->fetch(PDO::FETCH_BOUND);
        $count++;
    }
    printf("</pre>");

    } catch (PDOException $e) {
        printf("<pre>Error!: %s \n</pre>",$e->getMessage()) ;
        exit;
    }
    ?>

```

---

Using the `PDO::FETCH_CLASS` parameter to fetch a row into a new class instance is different from the use of parameters shown in Example 5-48 on page 248. You need to define the class name you want to use for the fetch with a `setFetchMode` call before you fetch the first row from the result set. Any attempts to specify the class name in the fetch call as you can in the `fetchAll` call, as shown in Example 5-53 on page 257, will return an error.

Example 5-49 on page 251 shows how to use `PDO::FETCH_CLASS` to fetch rows into several classes using a different type of properties. If you look at the output of the script, the fetch will simply use the `__set` method to assign the values to the properties of the specified class.

```
<?php

try {
$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");
$stmt=$dbh->query('SELECT t_type,t_model,t_price from car where t_id>1000');

/*
define fixed properties
*/
class car_fix
{
public $T_ID                ;
public $T_TYPE              ;
public $T_MODEL             ;
public $T_YEAR              ;
public $T_COLOR             ;
public $T_PACKAGE          ;
public $T_AC                ;
public $T_POWERWINDOWS     ;
public $T_AUTOMATIC         ;
public $T_DOORS             ;
public $T_PRICE             ;
public $T_DATA              ;
}
/*
define a variable class
*/
class car_variable
{
    function __set ($field, $value) {
        $this->$field=$value ;
    }
};
/*
define an array for the attributes
*/
class car_array
{
public $attributes=array();

    function __set ($field, $value) {
        $this->attributes[$field]=$value ;
    }
};
```

```

$stmt->setFetchMode(PDO::FETCH_CLASS,"car_variable");
$inst_car=$stmt->fetch();
print_r($inst_car);

$stmt->setFetchMode(PDO::FETCH_CLASS,"car_fix");
$inst_car=$stmt->fetch();
print_r($inst_car);

$stmt->setfetchmode(PDO::FETCH_CLASS,"car_array");
$inst_car=$stmt->fetch();
print_r($inst_car);

} catch (PDOException $e) {
    printf("<pre>Error!: %s \n</pre>", $e->getMessage());
    exit;
}
?>

```

The output of the script looks like this:

```

car_variable Object
(
    [T_TYPE] => Van
    [T_MODEL] => Primia
    [T_PRICE] => 16889.00
)
car_fix Object
(
    [T_ID] =>
    [T_TYPE] => Minivan
    [T_MODEL] => Boran
    [T_YEAR] =>
    [T_COLOR] =>
    [T_PACKAGE] =>
    [T_AC] =>
    [T_POWERWINDOWS] =>
    [T_AUTOMATIC] =>
    [T_DOORS] =>
    [T_PRICE] => 94883.00
    [T_DATA] =>
)
car_array Object
(
    [attributes] => Array
        (
            [T_TYPE] => SUV
            [T_MODEL] => Primia
            [T_PRICE] => 24429.00
        )
)

```

)

The standard cursor allows you to bring the data into various formats. However, you can only fetch in forward order. To allow result set navigation in variable directions, a scroll cursor should be used. The navigation capability does have an additional cost. Scroll cursors in Informix IDS are maintained by temporary tables. Opening a scroll cursor requires additional disk space in one of the temporary dbspaces. Example 5-50 shows the scroll cursor usage by modifying the PHP script in Example 5-48 on page 248.

---

*Example 5-50 Scroll cursors and Informix PDO*

---

```
<?php

try {
$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp; EnableScrollableCursors=1",
"informix", "123456");
if (!$dbh) { exit(); }

/*
static
*/

$stmt=$dbh->prepare('SELECT t_type,t_model,t_price from car where t_id=?',
array( PDO::ATTR_CURSOR => PDO::CURSOR_SCROLL ));
$stmt->execute(array(1500));

/*
5 rows into an assco array
*/
printf("<pre>\n");
printf("CARS with assoc arrays \n\n");
$row=$stmt->fetch(PDO::FETCH_ASSOC,PDO::FETCH_ORI_FIRST);
$count=0;
while ( $stmt && $count<5 && $row)
{
    printf(" %s %s %s \n",$row["T_TYPE"],$row["T_MODEL"],$row["T_PRICE"]);
    $row=$stmt->fetch(PDO::FETCH_ASSOC,PDO::FETCH_ORI_NEXT);
    $count++;
}
printf("</pre>");

/*
same 5 rows in reverse order into an numerated Array
*/
printf("<pre>\n");
```

```

printf("CARS with num arrays \n\n");
$row=$stmt->fetch(PDO::FETCH_NUM,PDO::FETCH_ORI_PRIOR);
$count=0;
while ( $stmt && $count<5 && $row)
{
    printf(" %s %s %s \n",$row[0],$row[1],$row[2]);
    $row=$stmt->fetch(PDO::FETCH_NUM,PDO::FETCH_ORI_PRIOR);
    $count++;
}
printf("</pre>");

printf("<pre>\n");
printf("CARS with an Object \n\n");
$row=$stmt->fetch(PDO::FETCH_OBJ,PDO::FETCH_ORI_LAST);
$count=0;
while ( $stmt && $count<5 && $row)
{
    printf(" %s %s %s \n",$row->T_TYPE,$row->T_MODEL,$row->T_PRICE);
    $row=$stmt->fetch(PDO::FETCH_OBJ, PDO::FETCH_ORI_REL , -11 );
    $count++;
}
printf("</pre>");

} catch (PDOException $e) {
    printf("<pre>Error!: %s \n</pre>",$e->getMessage()) ;
    exit;
}
?>

```

---

What are the differences in the implementation between a standard and a scroll cursor in Informix PDO?

You have to enable the scroll cursor support at connection time. There is an additional parameter `EnableScrollableCursors` in the DSN. Set the is parameter to 1 when you create a new instance of the PDO class.

- You have to specify the cursor type at prepare time. Positioning the cursor can be done within the fetch. You have several options of positioning the cursor, as shown in Example 5-50 on page 253. For making troubleshooting easier, Example 5-51 shows the error message about using scroll cursor when the DSN has not been enabled.

*Example 5-51 Error message when connecting without scroll cursors*

---

**PHP code fragment:**

```

$row=$stmt->fetch(PDO::FETCH_ASSOC,PDO::FETCH_ORI_FIRST);
print_r(($error=$dbh->errorInfo()));

```



**Output:**

```

Array
(
    [0] => HY106
    [1] => -11086
    [2] => [Informix][Informix ODBC Driver]Fetch type out of range.
(SQLFetchScroll[-11086] at
/home/holgerk/php/php-5.1.2.pdo/ext/pdo_informix/informix_statement.c:699)

```

---

Similar to the scroll cursor, an additional DSN parameter is required at connection time to work with cursors with hold. There is no need to specify the cursor type at the preparation time. Example 5-52 shows the code example of how to enable cursors with hold and where the hold cursors have different behavior than standard cursors.

*Example 5-52 CursorBehavior, DSN parameter, and the cursor in transactions***PHP script**

```

<?php

try {
$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp;CursorBehavior=1;", "informix",
"123456");
if (!$dbh) { exit(); }

/*
static
*/

$stmt=$dbh->prepare('SELECT t_type,t_model,t_price FROM car WHERE t_id?');
$stmt->execute(array(1500));
/*
5 rows into an assoc array
*/

$dbh->beginTransaction();
printf("<pre>\n");
printf("CARS with assoc arrays \n\n");
$row=$stmt->fetch(PDO::FETCH_ASSOC);
$count=0;
while ( $stmt && $count<5 && $row)
{
    printf(" %s %s %s \n",$row["T_TYPE"],$row["T_MODEL"],$row["T_PRICE"]);
    $row=$stmt->fetch(PDO::FETCH_ASSOC);
    $count++;
}

```

```

printf("</pre>");

$dbh->commit();

/*
Behaviour here different depending on which value is set for CursorBehavior
*/
printf("<pre>\n");
printf("CARS with num arrays \n\n");
$row=$stmt->fetch(PDO::FETCH_NUM);
$error=$dbh->errorInfo();
print_r($error);
$count=0;
while ( $stmt && $count<2 && $row)
{
    printf(" %s %s %s \n",$row[0],$row[1],$row[2]);
    $row=$stmt->fetch(PDO::FETCH_NUM);
    $count++;
}
printf("</pre>");

} catch (PDOException $e) {
    printf("<pre>Error!: %s \n</pre>",$e->getMessage()) ;
    exit;
}
?>

```

#### **Output with the CursorBehavior=1;:**

```

<pre>
CARS with assoc arrays

Van Boran 64013.00
Pickup UFIA 6791.00
Coupe Primia 36098.00
Pickup Garaunt 22727.00
Sedan Temia 78405.00
</pre><pre>
CARS with num arrays

Array
(
    [0] => 00000
    [1] => 0
    [2] => ((null)[0] at (null):0)
)
Coupe UFIA 48572.00
Minivan Temia 29904.00

```

```
</pre>
```

#### Output without the CursorBehavior=0,:

```
<pre>
CARS with assoc arrays

    Van Boran 64013.00
    Pickup UFIA 6791.00
    Coupe Primia 36098.00
    Pickup Garaunt 22727.00
    Sedan Temia 78405.00
</pre><pre>
CARS with num arrays

Array
(
    [0] => HY010
    [1] => -11067
    [2] => [Informix][Informix ODBC Driver]Function sequence error.
(SQLFetchScroll[-11067] at
/home/holgerk/php/php-5.1.2.pdo/ext/pdo_informix/informix_statement.c:699)
)
</pre>
```

---

Beside fetching the data one row at a time, the entire result set can be fetched into one array with one function call. This technique is not practical unless you know exactly how many rows will be returned and the size of result set is not too large. In Example 5-53, the data is fetched into the array at one time but we limited the number of rows fetched with the `FIRST N` clause.

#### *Example 5-53 First 10 rows and fetchAll in Informix PDO*

---

```
<?php
try {
    $dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");
    if (!$dbh) { exit(); }
    /*
    static
    */

    $stmt=$dbh->query('SELECT FIRST 10 t_type,t_model,t_price FROM car WHERE
t_id>1000');
    /*
    5 rows into an assoc array with fetchAll
    */
```

```

        printf("<pre>\n");
        printf("First 10 CARS into assco arrays with one fetch\n\n");
        $rows=$stmt->fetchAll(PDO::FETCH_ASSOC);
        print_r($rows);

/*
fetchAll and PDO::FETCH_CLASS
*/

class car_array
{
    public $attributes=array();

        function __set ($field, $value) {
            $this->attributes[$field]=$value ;
        }
};
    $stmt=$dbh->query('SELECT first 10 t_type,t_model,t_price from car where
t_id>1000');
    $rows=$stmt->fetchAll(PDO::FETCH_CLASS,"car_array");
    print_r($rows);

} catch (PDOException $e) {
    printf("<pre>Error!: %s \n</pre>", $e->getMessage()) ;
    exit;
}
?>

```

---

## Selects with ifx\_\* functions

Example 5-54 shows how to use the standard cursor in ifx\_\* . Keep in mind that ifx\_\* does not support dynamic SQL so you have to build your query before executing the SELECT.

*Example 5-54 Standard cursor with ifx\_\**

```

<?php

$link = ifx_connect("vps@on1000UC3soc","informix","123456");
if ( !$link ) {
    printf("<pre>");
    printf("Connect: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
    printf("</pre>");
    exit();
}

/*
simple query with output into assoc Array

```

```

*/
$statement='SELECT c_last,c_street,c_city FROM customer WHERE c_province
MATCHES "Sask*"';
$stmt=ifx_prepare($statement,$link);
ifx_do($stmt);

while ( ($row=ifx_fetch_row($stmt)))
{
    printf(" %s %s %s \n",$row["c_last"],$row["c_street"],$row["c_city"]);
}

ifx_free_result($stmt);

$statement='SELECT c_last,c_first,c_street from customer where c_city =
"Vancouver"';

$stmt=ifx_query($statement,$link);
ifx_htmltbl_result($stmt);
ifx_free_result($stmt);

@ifx_close($link);
?>

```

---

The result set will be returned as an associative array as shown in the body of the while loop. There is no function to return the row to PHP in a different format.

A similar way to get all results of a query is the use of `ifx_htmltbl_result` which will create a HTML table with all results.

`ifx_*` also provides a way to specify a scroll cursor at preparation time. Unlike the PDO, there is no need to specify that at connection time. With the scroll cursor, you can navigate the result set back and forth using with an additional parameter in `ifx_fetch_row`. You also can specify an offset in the result set. See Example 5-55.

---

*Example 5-55 Scroll cursors in ifx\_\**

---

```

<?php

$link = ifx_connect("vps@on1000UC3soc","informix","123456");
if ( !$link ) {
    printf("<pre>");
    printf("Connect: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
    printf("</pre>");
    exit();
}

/*

```

```

simple query with output into assoc Array
*/

$statement='SELECT c_last,c_street,c_city FROM customer WHERE c_province
MATCHES "Sask*";
$stmt=ifx_prepare($statement,$link,IFX_SCROLL);
ifx_do($stmt);

$count=1;
printf("The first 5 rows from the resultset \n\n");
while ( ($row=ifx_fetch_row($stmt,"NEXT")) && ($count < 5))
{
    printf(" %s %s %s \n",$row["c_last"],$row["c_street"],$row["c_city"]);
    $count++;
}

$count=1;
$row=ifx_fetch_row($stmt,"LAST");
printf("\n\nThe last 5 rows in reverse order \n\n");
while ( $row && ($count < 5))
{
    printf(" %s %s %s \n",$row["c_last"],$row["c_street"],$row["c_city"]);
    $count++;
    $row=ifx_fetch_row($stmt,"PREVIOUS");
}

$count=1;
$row=ifx_fetch_row($stmt,100);
printf("\nprint the 100 up to the 95 th row from the resultset \n\n");
while ( $row && ($count < 5))
{
    printf(" %s %s %s \n",$row["c_last"],$row["c_street"],$row["c_city"]);
    $count++;
    $row=ifx_fetch_row($stmt,"PREVIOUS");
}
ifx_free_result($stmt);
@ifx_close($link);
?>

```

---

Cursor with hold is also supported in ifx\_\*. Example 5-56 shows how to use cursor with hold. The same code example without IFX\_HOLD in the prepare would return an SQL code -400 at the next fetch after the COMMIT WORK.

---

*Example 5-56 Using cursor with hold in PHP with ifx\_\**

---

```

$link = ifx_connect("vps@on1000UC3soc","informix","123456");
if ( !$link ) {
    printf("<pre>");
    printf("Connect: Returncode %d -- %s \n",ifx_error(), ifx_errormsg());
}

```

```

        printf("</pre>");
        exit();
    }

    /*
    simple query with output into assoc Array and CUSROR WITH HOLD
    */

    $statement='SELECT c_last,c_street,c_city from customer where c_province
    matches "Sask*";
    $stmt=ifx_prepare($statement,$link,IFX_HOLD);
    ifx_do($stmt);

    ifx_query("BEGIN WORK",$link); /* simulate a transaction */
    $count=1;
    printf("The first 5 rows from the resultset \n\n");
    while ( ($row=ifx_fetch_row($stmt)) && ($count < 5))
    {
        printf(" %s %s %s \n",$row["c_last"],$row["c_street"],$row["c_city"]);
        $count++;
    }
    ifx_query("COMMIT WORK",$link); /* close the transaction */

    $count=1;
    $row=ifx_fetch_row($stmt);
    /* Error -400 here without IFX_HOLD */
    printf("\n\nThe next 5 rows \n\n");
    while ( $row && ($count < 5))
    {
        printf(" %s %s %s \n",$row["c_last"],$row["c_street"],$row["c_city"]);
        $count++;
        $row=ifx_fetch_row($stmt);
    }
    @ifx_close($link);
    ?>

```

---

## Selects with the PEAR DB package

The PEAR DB package enhances the capability of fetching data into associative arrays as we have seen for the `ifx_*` functions. You also can fetch the data into objects and numbered arrays. In Example 5-57 on page 262, we query the `PROMOTION` table to show the different ways to get the data into the PHP application. PEAR DB provides two ways to specify in which data structure the row will be placed. The `setFetchMode` method is used to specify the default data structure for all new cursors. The `fetchRow` method can specify, on the fly, the data structure for the current cursor.

getRow method returns only the first row of the result set. The result set will be freed after that. This is similar to using SELECT FIRST 1 in an SQL statement.

*Example 5-57 Using simple cursors in the DB package*

---

```
<?php
ini_set("include_path","/usr/local/lib/php");
require_once("DB.php");

$dbh = DB::connect("ifx://informix:123456@on1000UC3soc/vps");

$today =date("m/d/y");
$data = array($today,$today );

/* Simple Query with different FetchModes */

$stmt = $dbh->prepare('SELECT f_v_id,p_amount,p_start_date,p_end_date,p_desc
FROM promotion WHERE p_start_date<? AND p_end_date > ?;');
$res= $dbh->execute($stmt,$data);

$row=$res->fetchRow(DB_FETCHMODE_ORDERED);
if ($row) {
    printf("%s %s %s %s %s \n", $row[0], $row[1],$row[2],$row[3],$row[4]);
}
$row=$res->fetchRow(DB_FETCHMODE_ASSOC);
if ($row) {
    printf("%s %s %s %s %s \n", $row["f_v_id"], $row["p_amount"],
    $row["p_start_date"], $row["p_end_date"], $row["p_desc"]);
}
$row=$res->fetchRow(DB_FETCHMODE_OBJECT);
if ($row) {
    printf("%s %s %s %s %s \n", $row->f_v_id, $row->p_amount,
    $row->p_start_date, $row->p_end_date, $row->p_desc);
}
$res->free();

printf("\n\nOther Query result \n\n");

/*
Change the default FetchMode from DB_FETCHMODE_ASSOC to DB_FETCHMODE_OBJECT
*/

$dbh->setFetchMode(DB_FETCHMODE_OBJECT);

$stmt = $dbh->prepare('SELECT f_v_id,p_amount,p_start_date,p_end_date,p_desc
FROM promotion WHERE p_start_date<? AND p_end_date > ?;');
$res= $dbh->execute($stmt,$data);
```



```

$row=$res->fetchRow();
if ($row) {
    printf("%s %s %s %s %s \n", $row->f_v_id, $row->p_amount,
        $row->p_start_date, $row->p_end_date, $row->p_desc);
}
$res->free();

/*
get the first row from the resultset and free the resultset after than
*/

$row = $dbh->getRow('SELECT f_v_id,p_amount,p_start_date,p_end_date,p_desc FROM
promotion WHERE p_start_date<? AND p_end_date > ?;', $data);

if ($row) {
    printf("%s %s %s %s %s \n", $row->f_v_id, $row->p_amount,
        $row->p_start_date, $row->p_end_date, $row->p_desc);
}
?>

```

---

Scroll cursor is the only cursor used in PEAR DB. There is one limitation using scroll cursors with the PEAR DB package. You can use the offset in kind of a number. But since the total number of the result set is unknown, you will not be able to apply the offset to get to the last row. Example 5-58 demonstrates using the offsets in the fetchRow.

---

*Example 5-58 Using offsets for navigation in the result set in the DB package*

---

```

ini_set("include_path", "/usr/local/lib/php");
require_once("DB.php");

$dbh = DB::connect("ifx://informix:123456@on1000UC3soc/vps");

$today = date("m/d/y");
$data = array($today, $today );

/* Simple Query with different FetchModes */

$stmt = $dbh->prepare('SELECT f_v_id,p_amount,p_start_date,p_end_date,p_desc
FROM promotion WHERE p_start_date<? AND p_end_date > ?;');
$res= $dbh->execute($stmt,$data);

$row=$res->fetchRow(DB_FETCHMODE_ORDERED,2);
if ($row) {

```

```

        printf("%s %s %s %s %s \n", $row[0], $row[1], $row[2], $row[3], $row[4]);
    }
    $row=$res->fetchRow(DB_FETCHMODE_ASSOC,1);
    if ($row) {
        printf("%s %s %s %s %s \n", $row["f_v_id"], $row["p_amount"],
            $row["p_start_date"], $row["p_end_date"], $row["p_desc"]);
    }
    $row=$res->fetchRow(DB_FETCHMODE_OBJECT,4);
    if ($row) {
        printf("%s %s %s %s %s \n", $row->f_v_id, $row->p_amount,
            $row->p_start_date, $row->p_end_date, $row->p_desc);
    }
    $res->free();
?>

```

---

PEAR DB does not support the hold cursor. The application has to handle the parallel use of transactions and cursors.

But there are differences in understanding what a transaction is and is not. That is why we have here two examples for a better illustration, but we will only present the relevant parts from the code. For a detailed discussion about transactions, refer to 5.2.8, “Transactions and isolation level” on page 325.

Example 5-59 shows the usage of the native SQL transaction statements, the first fetch after a commit or rollback will return an error -400.

---

*Example 5-59 Native SQL transaction interface and simple cursors in PEAR DB*

---

```

$dbh->query("BEGIN WORK;");
/* Simple Query */

$stmt = $dbh->prepare('SELECT f_v_id,p_amount,p_start_date,p_end_date,p_desc
FROM promotion where p_start_date<? and p_end_date > ?;');
$res= $dbh->execute($stmt,$data);

$row=$res->fetchRow(DB_FETCHMODE_ORDERED,2);
if ($row) {
    printf("%s %s %s %s %s \n", $row[0], $row[1], $row[2], $row[3], $row[4]);
}
$dbh->query("COMMIT WORK;");
$row=$res->fetchRow(DB_FETCHMODE_ASSOC);
/* Error -400 here */

```

---

Example 5-60 on page 265 shows the same code with the autocommit interface provided by the PEAR DB package. Here, the second fetch will not fail even if the commit was issued. The reason is simply that no real transaction activity was done as far as changing data in the database, so the commit was not really

issued. Looking further in the example, a create table was executed, followed by a commit. In this case, a real transaction has happened (from the perspective of the DB), and the next fetch will fail with error code -400 (as we have already expected for the first fetch).

---

*Example 5-60 Using the autocommit interface with cursors*

---

```
$dbh->autoCommit(0);
/* Simple Query with different FetchModes */
$stmt = $dbh->prepare('SELECT f_v_id,p_amount,p_start_date,p_end_date,p_desc
FROM promotion where p_start_date<? and p_end_date > ?;');
$res= $dbh->execute($stmt,$data);

$row=$res->fetchRow(DB_FETCHMODE_ORDERED,2);
if ($row) {
    printf("%s %s %s %s %s \n", $row[0], $row[1], $row[2], $row[3], $row[4]);
}
$dbh->commit();
$row=$res->fetchRow(DB_FETCHMODE_ASSOC);
/* no error here !! */
if ($row) {
    printf("%s %s %s %s %s \n", $row["f_v_id"], $row["p_amount"],
    $row["p_start_date"], $row["p_end_date"], $row["p_desc"]);
}
$dbh->query("create table zz ( aa int)");
$dbh->commit();
$row=$res->fetchRow(DB_FETCHMODE_OBJECT);
/* But error -400 here !! */
if ($row) {
    printf("%s %s %s %s %s \n", $row->f_v_id, $row->p_amount,
    $row->p_start_date, $row->p_end_date, $row->p_desc);
}
}
```

---

## Selects with unixODBC

unixODBC supports all dynamic statements which return only a status, not a result set. Using the PREPARE and EXEC for dynamic, SELECT query will be executed, but there is not function to receive data. So we focus on static queries. Example 5-61 shows how to use the cursor to retrieve data into various data structures in unixODBC. You can get the data as an object or as an associative array. The function `odbc_fetch_row` only returns the row virtually. The real read on the row is done with `odbc_result`.

---

*Example 5-61 unixODBC and cursors*

---

```
<?php
```

```

$cn =
odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=vps","informix","123
456"
);
if ( !$cn ) {
    printf("<pre>Connect: attempt to connect was not successful \n</pre>");
    exit();
}

$str="SELECT * FROM neworder WHERE r_finalprice > 80000 ORDER BY 1";
$resultset=odbc_exec($cn,$str);

/*
fetching the row as an object
*/
$row=odbc_fetch_object($resultset);
if ($row) {
    printf("%s %s %s %s %s \n", $row->f_o_id, $row->r_price,
        $row->r_tax, $row->r_labour, $row->r_finalprice);
}
/*
fetching the row as an assoc array
*/

$row=odbc_fetch_array($resultset);
if ($row) {
    printf("%s %s %s %s %s \n", $row["f_o_id"], $row["r_price"],
        $row["r_tax"], $row["r_labour"], $row["r_finalprice"]);
}

/*
fetching the row virtually , work on the row with odbc_result after than
*/
odbc_fetch_row($resultset);
printf(" %s %s %s \n",odbc_result($odbc_result,1), odbc_result($odbc_result,2),
odbc_result($odbc_result,3));

odbc_free_result($resultset);

@odbc_close($cn);
?>

```

---

You also can retrieve the entire result set into an HTML table. Without knowing the size of the data, this can create performance problems. Example 5-62 on page 267 shows how to retrieve the entire result set. We limited the result to the first ten rows.

*Example 5-62 Returning all rows in unixODBC in an HTML table*

---

```
<?php
$cn =
odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=vps","informix","123
456"
);

$str="SELECT FIRST 10 * FROM neworder WHERE r_finalprice > 80000 ";
$ojdbc_result=odbc_exec($cn,$str);

odbc_result_all($ojdbc_result);
odbc_free_result($ojdbc_result);

@odbc_close($cn);
?>
```

---

unix ODBC supports scroll cursor. To enable the scroll cursor, you need to specify an additional parameter in the DSN at connection time. Example 5-63 provides a complete example of how to enable and use a scroll cursor.

*Example 5-63 Using scroll cursors in unixODBC*

---

```
<?php
$cn =
odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=vps;EnableScrollable
Cursors=1","informix","123456"
);
$str="SELECT * FROM neworder WHERE r_finalprice > 80000 ORDER BY 1";
$ojdbc_result=odbc_exec($cn,$str);
/*
Using offsets
*/

$row=odbc_fetch_object($ojdbc_result,1);
if ($row) {
    printf("%s %s %s %s %s \n", $row->f_o_id, $row->r_price,
        $row->r_tax, $row->r_labour, $row->r_finalprice);
}
$row=odbc_fetch_array($ojdbc_result,30);
if ($row) {
    printf("%s %s %s %s %s \n", $row["f_o_id"], $row["r_price"],
        $row["r_tax"], $row["r_labour"], $row["r_finalprice"]);
}
$row=odbc_fetch_object($ojdbc_result,20);
if ($row) {
    printf("%s %s %s %s %s \n", $row->f_o_id, $row->r_price,
        $row->r_tax, $row->r_labour, $row->r_finalprice);
}
}
```

```

$row=odbc_fetch_array($odbc_result,22);
if ($row) {
    printf("%s %s %s %s %s \n", $row["f_o_id"], $row["r_price"],
        $row["r_tax"], $row["r_labour"], $row["r_finalprice"]);
}
odbc_free_result($odbc_result);
@odbc_close($cn);
?>

```

---

In unixODBC, cursors with hold are enabled by the `CursorBehavior` parameter in the DSN. If this parameter is set to 1, ODBC will open the cursors with hold. If this value is a 0, which is the default, the cursor will be a standard cursor.

Example 5-64 shows the usage of the hold cursor. In this example, after fetching two rows and executing the commit, the cursor will remain open. Rows 3 and 4 will be fetched after that. If standard cursor is declared, the cursor is closed by the commit after the second row is fetched. Only two rows will be returned.

---

*Example 5-64 Using a hold cursor with unixODBC*

---

**PHP Script:**

```

<?php

$cn =
odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=vps;CursorBehavior=1
;", "informix", "123456"
);
if ( !$cn ) {
    printf("<pre>Connect: attempt to connect was not successful \n</pre>");
    exit();
}

odbc_autocommit($cn,FALSE);
/* for using the SQL interface run
odbc_exec($cn,"begin work;");
*/
$str="select * from neworder where r_finalprice > 10000 ";
$odbc_result=odbc_exec($cn,$str);

$row=odbc_fetch_object($odbc_result);
if ($row) {
    printf("1: %s %s %s %s %s \n", $row->f_o_id, $row->r_price,
        $row->r_tax, $row->r_labour, $row->r_finalprice);
}
$row=odbc_fetch_array($odbc_result);
if ($row) {
    printf("2: %s %s %s %s %s \n", $row["f_o_id"], $row["r_price"],
        $row["r_tax"], $row["r_labour"], $row["r_finalprice"]);
}

```

```

}
/* for using the SQL interface run
odbc_exec($cn,"commit work;");
*/
odbc_commit($cn);

$row=odbc_fetch_array($odbc_result);
if ($row) {
    printf("3: %s %s %s %s %s \n", $row["f_o_id"], $row["r_price"],
        $row["r_tax"], $row["r_labour"], $row["r_finalprice"]);
}

$row=odbc_fetch_array($odbc_result);
if ($row) {
    printf("4: %s %s %s %s %s \n", $row["f_o_id"], $row["r_price"],
        $row["r_tax"], $row["r_labour"], $row["r_finalprice"]);
}

odbc_free_result($odbc_result);
@odbc_close($cn);
?>

```

#### **Output:**

with hold cursor ( 4 rows)

```

1: 1 86367.00 1.00 917.00 87284.00
2: 2 62654.00 1.00 150.00 62804.00
3: 3 86583.00 2.00 293.00 86876.00
4: 4 79373.00 8.00 875.00 80248.00

```

without hold cursor ( 2 rows)

```

1: 1 86367.00 1.00 917.00 87284.00
2: 2 62654.00 1.00 150.00 62804.00

```

---

## **5.2.4 Complex data types**

In this section, we discuss how to use complex data types in a PHP program. We cover row types and collection types, such as SET and LIST. We explain how to work with complex data types using examples with Informix PDO.

### **Named row types**

If you take a close look at the CUSTOMER and the DEALER tables in our Dealership database, you will noticed that both tables have a lot of columns which have the same meaning. Besides name differences, definition and usage of column are the same. We redefine these two tables using row type for these columns. See

Example 5-65. Name and the address columns are now defined by a row type “address”.

*Example 5-65 Using row types in Informix IDS database tables*

---

```
create row type 'informix'.address
(
    d_name varchar(50) not null ,
    d_street varchar(50),
    d_city varchar(50),
    d_province varchar(20),
    d_pcode varchar(10)
);
create table "vps".dealer_with_rowtype
(
    d_id serial not null ,
    f_w_id integer not null ,
    f_l_id integer not null ,
    d_address address,
    d_stax decimal(3,2),
    d_ytd decimal(20,2),
    unique (d_id)
);
create table "vps".customer_with_rowtype
(
    c_id serial not null ,
    c_sal varchar(10) not null ,
    c_address address,
    c_phone varchar(20),
    c_creditlimit decimal(10,2),
    c_balance decimal(20,2),
    c_ytdpayment decimal(20,2),
    c_data lvarchar(500)
);
```

---

The PHP script in Example 5-66 shows SQL statements using row type. This script inserts a dealer and two customers, then selects the customer data from the database with and without a filter on the row type. It then updates one of the customers and delete the rows from the table. At the bottom of the example, we provide the output of one row from the select to give you an idea about how the array with row types looks in PHP.

*Example 5-66 Using named row types in Informix PDO*

---

```
<?php

$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");
if (!$dbh) {
```



```

        exit();
    }

    printf(" ***** Insert ***** \n\n");

    /*
    Insert a row with a rowtype -- add a dealer and 2 customers
    open the car buisness
    */
    $dbh->query(' INSERT INTO dealer_with_rowtype VALUES (
    0,7,24,row("Ridgewater","Broadway","Winnipeg","Alberta","CODE")::address, 1,
    186111)');
    print_r(($error=$dbh->errorInfo()));
    $dbh->query(' INSERT INTO customer_with_rowtype VALUES (
    0,"Mrs.",row("Smith","Queen","Ottawa","Newfoundland","CODE")::address,5143259,
    2678876,15263390,8719,"")');
    $dbh->query(' INSERT INTO customer_with_rowtype VALUES (
    0,"Mrs.",row("Miller","Queen","Ottawa","Newfoundland","CODE")::address,5143259,
    2678876,15263390,8719,"")');

    printf(" ***** Select woF ***** \n\n");
    /*
    select the customer row without any filters
    */

    $stmt1=$dbh->query(" SELECT c_id , c_sal , c_address::lvarchar, c_phone ,
    c_creditlimit , c_balance , c_ytdpayment , c_data FROM customer_with_rowtype
    ");
    $row=$stmt1->fetch(PDO::FETCH_ASSOC);
    while($row) {
        print_r($row);
        $row=$stmt1->fetch(PDO::FETCH_ASSOC);
    }

    printf(" ***** Select wF ***** \n\n");
    /*
    select the row with a row_type filter ( a given address )
    */

    $stmt1=$dbh->query(' SELECT c_id , c_sal , c_address::lvarchar, c_phone ,
    c_creditlimit , c_balance , c_ytdpayment , c_data FROM customer_with_rowtype
    WHERE row("Smith","Queen","Ottawa","Newfoundland","CODE")::address =
    c_address');
    print_r(($error=$dbh->errorInfo()));
    $row=$stmt1->fetch(PDO::FETCH_ASSOC);
    while($row) {
        print_r($row);
        $row=$stmt1->fetch(PDO::FETCH_ASSOC);
    }
}

```

```

printf(" ***** Update ***** \n\n");
/*
customer moved -- update customer address
*/

$stmt1=$dbh->query(' UPDATE customer_with_rowtype SET
c_address=row("Smith","Kings","Toronto","Manitoba","CODE")::address WHERE
row("Smith","Queen","Ottawa","Newfoundland","CODE")::address = c_address');

printf(" ***** SELECT ***** \n\n");
/*
Verification
*/
$stmt1=$dbh->query(" SELECT c_id , c_sal , c_address::lvarchar, c_phone ,
c_creditlimit , c_balance , c_ytdpayment , c_data FROM customer_with_rowtype
");
$row=$stmt1->fetch(PDO::FETCH_ASSOC);
while($row) {
    print_r($row);
    $row=$stmt1->fetch(PDO::FETCH_ASSOC);
}

printf(" ***** DELETE ***** \n\n");
/*
close the buisness
*/

$dbh->query(' DELETE FROM dealer_with_rowtype ');
$dbh->query(' DELETE FROM customer_with_rowtype ');

?>

```

**Output of the script for one customer:**

```

Array
(
    [C_ID] => 9
    [C_SAL] => Mrs.
    [] => ROW('Miller','Queen','Ottawa','Newfoundland','CODE')
    [C_PHONE] => 5143259
    [C_CREDITLIMIT] => 2678876.00
    [C_BALANCE] => 15263390.00
    [C_YTDPAYMENT] => 8719.00
    [C_DATA] =>
)

```

---

For the explanation about why the field name in the output of the associative array for the row type is "", see the discussion after Example 5-67.

## Collection types

The collection data types are another set of complex data types provided by the Informix IDS database server. Collection data types include set, list, and multiset. In this section, we cover the list and set. We also show a combination of the collection data type with the unnamed row type which is similar to the named row type.

The CAR table has several attributes describing a car. Color is one of the attributes. You may like to have all the colors available for a car included in the CAR table in one column. To make this scenario more complicated, we combine the CAR table and the CARLOT table together. We define the car lot, which is the garage in which a car is placed, as a list in one column in the CAR table. Example 5-67 shows the schema of the CAR table using the set and the list data types.

*Example 5-67 Collection data types and unnamed row types in a table schema*

---

```
CREATE TABLE car_multiset
(
    t_id serial not null ,
    t_type varchar(50) not null ,
    t_model varchar(50) not null ,
    t_year integer,
    t_color SET(VARCHAR(30) NOT NULL),
    t_package varchar(50),
    t_ac char(1),
    t_powerwindows char(1),
    t_automatic char(1),
    t_doors smallint,
    t_price decimal(10,2) not null ,
    car_lot list (
        ROW(
            l_street varchar(50),
            l_city varchar(50),
            l_province varchar(20),
            l_pcode varchar(10),
            l_stax decimal(3,2),
            l_ytd decimal(20,2)
        ) not null
    )
);
```

---

Example 5-68 shows how to use these complex data types in a PHP script, using this table as a base for Informix PDO functions. This script does the following:

- ▶ Inserts some cars into the table
- ▶ Selects cars without a filter condition
- ▶ Selects cars, using color, which is a set, as the filter
- ▶ Selects cars using car lot, which is a list, as the filter

**Note:** With Version 1.0.0 of Informix PDO, you have to cast complex data types to an LVARCHAR in a select statement, as seen in Example 5-68 and Example 5-66, to get the output. Since this cast is an expression, tools such as dbaccess would return as a column name the string “expression”. Using an associative array in Informix PDO, the column name will be empty for columns built up from expressions in the result. You have to rename the result column in the SELECT; otherwise, only the last expression column will be stored in the array. The reason is that you must have unique identifiers for the members of the array. If the same identifier is used twice, there will not be another member added, since the value for the already existing identifier will be overwritten. Another way to get all result columns is the use of a numbered array, where the array members will be identified by an integer, which is unique.

#### *Example 5-68 Using collections in Informix PDO*

```
<?php

$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456",
array(PDO::ATTR_PERSISTENT=> true));
if (!$dbh) {
    exit();
}

printf(" ***** Insert ***** \n\n");
/*
Insert a row with a rowtype
open the car buisness
*/
$dbbh->query(" INSERT INTO car_multiset VALUES ( 0,
'Minivan','Garaunt',1967,\"SET{'Red','Blue','Yellow'}\\",'N','N','N','Y',6,76323
, LIST{ ROW( 'Duplex','Montreal','Manitoba','CODE',6,109886), ROW(
'Palace','Regina','British Columbia','CODE',6,109886), ROW(
'Queen','Toronto','Manitoba','CODE',6,109886), ROW(
'Hallway','Vancouver','PEI','CODE',6,109886), ROW(
'King','Winnipeg','Newfoundland','CODE',6,109886)}}");

$dbbh->query(" INSERT INTO car_multiset VALUES ( 0,
'Coupe','Garaunt',1967,\"SET{'Green'}\\",'N','N','N','Y',6,76323, LIST{ ROW(
'Duplex','Montreal','Manitoba','CODE',6,109886), ROW( 'Cottle','San
Jose','CA','CODE',6,109886), ROW(
'Hollerith','Munich','Germany','CODE',6,109886)}}");
```

```

print_r(($error=$dbh->errorInfo()));

printf(" ***** Select woF ***** \n\n");
/*
select the row without any filters
*/

$stmt1=$dbh->query(" SELECT t_id, t_type, t_model, t_year, t_color::lvarchar
t_color1, t_package, t_ac, t_powerwindows, t_automatic, t_doors, t_price ,
car_lot::lvarchar car_lot1 FROM car_multiset");
print_r(($error=$dbh->errorInfo()));
$row=$stmt1->fetch(PDO::FETCH_ASSOC);
while($row) {
    print_r($row);
    $row=$stmt1->fetch(PDO::FETCH_ASSOC);
}
printf(" ***** Select wF ***** \n\n");
/*
select the row with a set filter for the color of the car
*/

$stmt1=$dbh->query(" SELECT t_id, t_type, t_model, t_year, t_color::lvarchar
t_color1, t_package, t_ac, t_powerwindows, t_automatic, t_doors, t_price ,
car_lot::lvarchar car_lot1 FROM car_multiset WHERE 'Red' IN t_color");
print_r(($error=$dbh->errorInfo()));
$row=$stmt1->fetch(PDO::FETCH_ASSOC);
while($row) {
    print_r($row);
    $row=$stmt1->fetch(PDO::FETCH_ASSOC);
}

printf(" ***** Select wF ***** \n\n");
/*
select the row with a set filter for the carlot
*/
$stmt1=$dbh->query(" SELECT t_id, t_type, t_model, t_year, t_color::lvarchar
t_color1, t_package, t_ac, t_powerwindows, t_automatic, t_doors, t_price ,
car_lot::lvarchar car_lot1 FROM car_multiset WHERE ROW( 'Cottle','San
Jose','CA','CODE',6,109886) in car_lot");

print_r(($error=$dbh->errorInfo()));
$row=$stmt1->fetch(PDO::FETCH_ASSOC);
while($row) {
    print_r($row);
    $row=$stmt1->fetch(PDO::FETCH_ASSOC);
}
printf(" ***** DELETE ***** \n\n");
/*

```

```

close the buisness
*/
$dbbh->query(' DELETE FROM car_multiset ');
?>

```

**Output of one row:**

```

Array
(
    [T_ID] => 15
    [T_TYPE] => Coupe
    [T_MODEL] => Garaunt
    [T_YEAR] => 1967
    [T_COLOR1] => SET{'Green'}
    [T_PACKAGE] => N
    [T_AC] => N
    [T_POWERWINDOWS] => N
    [T_AUTOMATIC] => Y
    [T_DOORS] => 6
    [T_PRICE] => 76323.00
    [CAR_LOT1] => LIST{ROW('Duplex','Montreal','Manitoba','CODE',6.00
,109886.00      ),ROW('Cottle','San Jose','CA','CODE',6.00 ,109886.00
),ROW('Hollerith','Munich','Germany','CODE',6.00 ,109886.00      )}}
)

```

---

## 5.2.5 Working with stored procedures and user defined functions

Whether or not to use stored procedures is a design decision that depends on whether the application can take advantage of the server-based source code or not. The considerations are:

- ▶ Performance and server utilization

This is the major consideration. Running most of the SQL code on the server is not only faster, but it also utilizes the database server resource more. The communication between the server and client application is also reduced, which improves the performance.

- ▶ Source code maintenance

Stored procedures are a server-side application which requires the same level of code maintenance as other applications, including good documentation and version management. In addition, communication between the database administrator and the application development team is necessary.

- ▶ Permissions

There is another level for object permissions. The execute permissions for stored procedures have to be granted and maintained.

The Informix IDS database server provides the ability to create stored procedures in the proprietary language Stored Procedure Language (SPL), Java, or C. Using the SPL, the stored procedures will be checked for syntactical correctness at creation time and a best possible plan, based on the statistics at creation time, will be created. So it could be understood as “pre-optimized”. The plan and the stored procedure will be stored in the database system catalog where the stored procedure is created. If the base table of the stored procedure has a DDL change or the table statistic is updated, the stored procedure plan will be automatically rebuilt at the next execution time.

For additional information about the stored procedures, refer to *IBM Informix SQL Tutorial*, GC12-3482, and *IBM Informix SQL Syntax Guide*, G251-2284-02.

There are differences between using a stored procedure, which returns one or more rows, and a stored procedure, which returns nothing, in the PHP script. In this section, we discuss both using various Informix database interfaces.

## Stored procedures with Informix PDO

Example 5-69 shows how to write and use stored procedures in the Informix IDS with Informix PDO. This is a simple new user registration application in our Dealership application. There are three routines in the example:

- ▶ User interface: A form for the user to enter information.
- ▶ User registration PHP script: This PHP script is called after the registration is submitted. The PHP script then calls the stored procedure to insert the user record.
- ▶ Stored procedure: The stored procedure simply inserts the user into the database table using a dynamic statement. It does not return any value.

This example certainly has room for improvement. The customer data validation can be added in the PHP script. Additional data verification in the database can include logical zip code verification to make sure that the zip code matches with the street and the city.

*Example 5-69 Register a user using a stored procedure with Informix PDO*

---

### 1. Simple user interface

```
<?php

printf("<form method=\"post\" action=\"reg_usersp.php\" >");
printf("Name <input name=\"name\" /><br />");
printf("Mail <input name=\"mail\" /><br />");
printf("Street <input name=\"street\" /><br />");
printf("City <input name=\"city\" /><br />");
printf("State <input name=\"state\" /><br />");
```

```

        printf("Zip <input name=\"zip\" /><br />");
        printf("Phone <input name=\"phone\" /><br />");
        printf("<input type=\"submit\" value=\"Confirm\" name=\"Button\"/>");
        printf("<input type=\"submit\" value=\"Abort\" name=\"Cancel\"/>");
        printf("</form>");
?>

```

## 2. PHP script - reg\_usersp.php

```

<?php

if (isset($_POST["Button"])) {

$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");

if (!$dbh) {
    exit(1);
}

$sph= $dbh->prepare(" EXECUTE PROCEDURE register_user ( 0,?,?,?,?,?,0);");
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" prepare procedure failed with %s \n",$error["1"]);
    exit(1);
}

$sph->bindParam(1, $_POST["name"],PDO::PARAM_STR, 20);
$sph->bindParam(2, $_POST["mail"],PDO::PARAM_STR, 20);
$sph->bindParam(3, $_POST["street"],PDO::PARAM_STR, 20);
$sph->bindParam(4, $_POST["city"],PDO::PARAM_STR, 20);
$sph->bindParam(5, $_POST["state"],PDO::PARAM_STR, 20);
$sph->bindParam(6, $_POST["zip"],PDO::PARAM_STR, 20);
$sph->bindParam(7, $_POST["phone"],PDO::PARAM_STR, 20);

$sph->execute();
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" execute sp failed with %s \n",$error["1"]);
    print_r($error);
    exit(1);
}
}
else {
    printf("Registration was cancelled \n");
}
?>

```

## 3. Stored procedure in the server - register\_user()



```

DROP PROCEDURE informix.register_user;
CREATE PROCEDURE register_user (
  c_id integer,
  c_name char(30),
  c_email char(30),
  c_street char(30),
  c_city char(30),
  c_state char(30),
  c_phone char(20),
  c_zip char(20),
  c_data char(30)
)
DEFINE eisam integer;
DEFINE esql integer;

--set debug file to "/tmp/register_user";
--trace on;

BEGIN
ON EXCEPTION SET esql, eisam
IF esql = -239 THEN -- duplicate value
RAISE EXCEPTION -746 , 0, "Customer Number already exists ";
END IF

END EXCEPTION

INSERT INTO customer VALUES (
  c_id, c_name, c_email, c_street, c_city, c_state, c_phone, c_zip, c_data);

END

end procedure;

```

---

In the stored procedure, we added a block of SPL for error handling. The error handling code is embraced in the BEGIN and END statements. The ON EXCEPTION SET assigns the SQL error to the predefined variables. The RAISE EXCEPTION gives us the ability to redefine the actual error code to a different number with a user-defined error message. In the example, we attempted to insert the same customer twice, error code -746. `errorInfo()` is the Informix PDO error handling function which can determine error codes and the appropriate action can be taken based on the error code. For an example of this, refer to the procedure `insert_orders` in Example 5-75 on page 297.

The stored procedure in Example 5-69 on page 277 inserts a user into a database and does not require any data returned to the PHP script. The SQL execution status is dealt with by the error handling code directly inside the stored procedure. The stored procedure also can return one or more rows in a result set to the calling routine. The stored procedures returning values back to the PHP script can be treated like simple selects. Both SQL statements SELECT and EXECUTE PROCEDURE need the existence of a cursor to navigate in the result set. Also, cursors with hold and scroll cursors could be applied to a result set of an EXECUTE PROCEDURE statement. To use scroll and hold cursors, you have to add parameters to the connection string. You have to specify a scroll cursor at preparation time. For further information about the parameter setting for both cursors, refer to “SELECT statements with Informix PDO” on page 248.

Example 5-70 shows how to work with stored procedures which return data to the calling routine. This example includes:

- ▶ An HTML form for the user to enter car searching criteria.
- ▶ A PHP program which shows how to call the stored procedure and access the data returned by the stored procedure.
- ▶ A stored procedure which takes the car search criteria as input and returns the cars found.

---

*Example 5-70 Returning a result set with stored procedures within Informix PDO*

**1. HTML**

```
<?php
printf("<form method=\"post\" action=\"search_carsp.php\" >");
printf("type <input name=\"type\" /><br />");
printf("model <input name=\"model\" /><br />");
printf("year <input name=\"year\" /><br />");
printf("color <input name=\"color\" /><br />");
printf("package <input name=\"package\" /><br />");
printf("AC <input name=\"ac\" /><br />");
printf("Windows <input name=\"powerwindows\" /><br />");
printf("Automatic <input name=\"automatic\" /><br />");
printf("Doors <input name=\"doors\" /><br />");
printf("Price <input name=\"price\" /><br />");
printf("Extras <input name=\"extra\" /><br />");
printf("<input type=\"submit\" value=\"Confirm\" name=\"Button\"/>");
printf("<input type=\"submit\" value=\"Abort\" name=\"Cancel\"/>");
printf("</form>");
?>
```

**2. PHP script - search\_carsp.php**

```
<?php
```

```

$DEBUG=1;

if (isset($_POST["Button"])) {

$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix",
"123456",array(PDO::ATTR_PERSISTENT=> true));

if (!$dbh) {
    exit(1);
}

$sph= $dbh->prepare(" EXECUTE PROCEDURE search_for_cars (
?,?,?,?,?,?,?,?,?,?);");
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" prepare procedure failed with %s \n",$error["1"]);
    exit(1);
}

$sph->bindParam(1, $_POST["type"],PDO::PARAM_STR, 20);
$sph->bindParam(2, $_POST["model"],PDO::PARAM_STR, 20);
$sph->bindParam(3, $_POST["year"],PDO::PARAM_STR, 20);
$sph->bindParam(4, $_POST["color"],PDO::PARAM_STR, 20);
$sph->bindParam(5, $_POST["package"],PDO::PARAM_STR, 20);
$sph->bindParam(6, $_POST["ac"],PDO::PARAM_STR, 20);
$sph->bindParam(7, $_POST["powerwindows"],PDO::PARAM_STR, 20);
$sph->bindParam(8, $_POST["automatic"],PDO::PARAM_STR, 20);
$sph->bindParam(9, $_POST["doors"],PDO::PARAM_INT);
$sph->bindParam(10, $_POST["price"],PDO::PARAM_INT);
$sph->bindParam(11, $DEBUG,PDO::PARAM_INT);

printf("<pre>");
$sph->execute();
$error=$dbh->errorInfo();
if ( $error["1"]) print_r ($error);

$row=$sph->fetch(PDO::FETCH_NUM);
while ( $row && $count < 20 && !$error["1"])
{
    print_r ( $row) ;
    $count ++;
    $row=$sph->fetch(PDO::FETCH_NUM);
    $error=$dbh->errorInfo();
}
$row=$sph->fetch(PDO::FETCH_ASSOC);
print_r ( $row) ;

if ( $error["1"]) print_r ($error);

```

```

printf("</pre>");

}
else {
    printf("Search for the cars was finished \n");
}
?>

```

### 3. Stored Procedure - search\_for\_cars

```

dbaccess << EOF
database vps;

```

```

DROP PROCEDURE search_for_cars;

```

```

CREATE PROCEDURE search_for_cars (
m_type varchar(50),
m_model varchar(50),
m_year varchar(4),
m_color varchar(50),
m_package varchar(50),
m_ac varchar(1),
m_power varchar(1),
m_automatic varchar(1),
m_doors integer,
m_price decimal(12,2),
DEBUG integer
) returning integer, varchar(30), varchar(30),integer,varchar(30),varchar(30),
varchar(30), varchar(20),varchar(20), integer, decimal(12,2);

```

```

define local_m_type char(50);
define local_m_model char(50);
define local_m_year integer;
define local_m_color char(50);
define local_m_package char(50);
define local_m_ac char(1);
define local_m_power char(1);
define local_m_automatic char(1);
define local_m_doors integer;
define local_m_price decimal(12,2);

```

```

DEFINE llocal_m_id integer;
DEFINE llocal_m_type char(50);
DEFINE llocal_m_model char(50);
DEFINE llocal_m_year integer;
DEFINE llocal_m_color char(50);
DEFINE llocal_m_package char(50);

```

```

DEFINE llocal_m_ac char(1);
DEFINE llocal_m_power char(1);
DEFINE llocal_m_automatic char(1);
DEFINE llocal_m_doors integer;
DEFINE llocal_m_price decimal(12,2);
DEFINE llocal_m_data char(30000);

IF DEBUG = 1 THEN
SET DEBUG FILE to "/tmp/search_for_cars";
TRACE ON;
SET EXPLAIN ON;
END IF

LET local_m_type = trim(trim(m_type) || "");
LET local_m_model = trim(trim(m_model) || "");
LET local_m_year = trim(m_year) ;
LET local_m_color = trim(trim(m_color) || "");
LET local_m_package = trim(trim(m_package) || "");
LET local_m_ac = trim(trim(m_ac) || "");
LET local_m_power = trim(trim(m_power) || "");
LET local_m_automatic = trim(trim(m_automatic) || "");

IF m_doors < 2 THEN LET local_m_doors = 2; ELSE LET local_m_doors= m_doors; END
IF
IF m_price < 1000 then let local_m_price = 10000.00; ELSE LET
local_m_price=m_price; END IF

FOREACH c1 FOR

SELECT *
INTO llocal_m_id , llocal_m_type , llocal_m_model , llocal_m_year,
llocal_m_color ,
llocal_m_package , llocal_m_ac , llocal_m_power , llocal_m_automatic ,
llocal_m_doors , llocal_m_price , llocal_m_data
FROM car
WHERE
t_type MATCHES local_m_type and
t_model MATCHES local_m_model and
t_year::int > local_m_year and
t_color MATCHES local_m_color and
t_package MATCHES local_m_package and
t_ac MATCHES local_m_ac and
t_powerwindows MATCHES local_m_power and
t_automatic MATCHES local_m_automatic and
t_doors = local_m_doors and
t_price < local_m_price

RETURN llocal_m_id,
llocal_m_type , llocal_m_model , llocal_m_year, llocal_m_color ,

```

```
llocal_m_package , llocal_m_ac , llocal_m_power , llocal_m_automatic ,
llocal_m_doors , llocal_m_price WITH RESUME;

END FOREACH;
END PROCEDURE;
```

---

You can use fetch to get the data to various data structures. We only demonstrate using the FETCH\_NUM. You should not use FETCH\_ASSOC for fetching result sets from a stored procedure in Informix PDO. FETCH\_ASSOC returns only the last column of the row in the array returned by fetch. If you look into dbaccess and run the similar stored procedure there, you will see that the headings of all columns are named "(expression)". In Informix PDO Version 1.0, this will be substituted by "". For a unique determination in an associative array, only one "" as a array field name can exist, each new occurrence for the same field name will only overwrite the value, not generate a new field. That is why you will only see the last column in that array.

If you plan, similar to our stored procedure example, to use local variables in the where clause of a statement, and in the into clause of the SELECT statement, use different variable names for that. This simple stored procedure could make the application programmer think it is alright to use only one set of variables for both, but the result is other than expected. You would only see one row in the result set all the time.

### ***Error tracing and error handling in stored procedures***

In a application development cycle, the debugging or tracing facilities are very helpful for finding the defects in the code. Since the stored procedure is also application code, you should consider adding debugging function in the stored procedure. In Example 5-70 on page 280, we added a DEBUG parameter which, if set to 1, turns the trace on. The PHP script calling the stored procedure can define if tracing is needed or not. The SPL statement TRACE ON writes stored procedure trace information to the file defined by SET DEBUG FILE on the database server machine. Example 5-71 shows a trace file sample.

#### ***Example 5-71 Trace file generated by a stored procedure***

---

```
trace on

set explain on;
expression:TRIM ( BOTH ' ' FROM (|| TRIM ( BOTH ' ' FROM m_type), "*" )
evaluates to Minivan*
let local_m_type = Minivan*
.....
evaluates to *
```

```

let local_m_automatic = *
expression:(< m_doors, 2)
evaluates to f
expression:m_doors
evaluates to 2
let local_m_doors = 2
expression:(< m_price, 1000)
evaluates to f
expression:m_price
evaluates to 10000000.00
let local_m_price = 10000000.00
start select cursor.
"0$c1" is select *
    from car
    where (and (and (and (and (and (and (and (and (and (matches t_type,
local_m_type), (matches t_model, local_m_model)), (> t_year::INT, 1000)),
(matches t_color, local_m_color)), (matches t_package, local_m_package)),
(matches t_ac, local_m_ac)), (matches t_powerwindows, local_m_power)), (matches
t_automatic, local_m_automatic)), (= t_doors, local_m_doors)), (< t_price,
local_m_price))
select cursor iteration.
.....

```

---

In the stored procedure, using local defined variables in the queries can influence the query plan. Use a `SET EXPLAIN ON` to analyze how a specific SQL statement is executed for an optimization of the procedure. Once the `SET EXPLAIN ON` is executed, it will generate a `sqexplain.out` file. The file contains the access plan of the statements for a user session running the stored procedure. This file is created in the home directory of the remote user on the database server, or in the local directory for a local user. If you want to verify that an index is used, examining the `sqexplain.out` file is a good starting point. Example 5-72 shows the `sqexplain.out` file of our `search_for_cars` stored procedure.

---

*Example 5-72 sqexplain for a stored procedure and their queries*

---

```

:
-----
Procedure: informix.search_for_cars
select x0.t_id ,x0.t_type ,x0.t_model ,x0.t_year ,x0.t_color ,x0.t_package
,x0.t_ac ,x0.t_powerwindows ,x0.t_automatic ,x0.t_doors ,x0.t_price ,x0.t_data
::char from "vps".car x0 where (((((((((x0.t_type MATCHES ? ) AND (x0.t_model
MATCHES ? ) ) AND (x0.t_year ::integer > 1000 ) ) AND (x0.t_color MATCHES ? ) )
AND (x0.t_package MATCHES ? ) ) AND (x0.t_ac MATCHES ? ) ) AND
(x0.t_powerwindows MATCHES ? ) ) AND (x0.t_automatic MATCHES ? ) ) AND
(x0.t_doors = ? ) ) AND (x0.t_price < ? ) )

QUERY:

```

-----

Estimated Cost: 49  
Estimated # of Rows Returned: 1

1) vps.car: INDEX PATH

```
Filters: ((((((vps.car.t_type MATCHES 'Minivan*
' AND vps.car.t_package MATCHES '*'
' ) AND vps.car.t_color MATCHES '*'
' ) AND vps.car.t_doors = 2 ) AND vps.car.t_year ::integer> 1999 ) AND
vps.car.t_automatic MATCHES '*' ) AND vps.car.t_powerwindows MATCHES '*' ) AND
vps.car.t_ac MATCHES '*' )
```

(1) Index Keys: t\_type t\_model t\_year t\_price (Key-First) (Serial,  
fragments: ALL)

Lower Index Filter: vps.car.t\_type MATCHES 'Minivan\*'

Key-First Filters: (vps.car.t\_model MATCHES '\*'  
' ) AND  
(vps.car.t\_price < 10000000.00 )

---

## Stored procedures with ifx\_\*

The example to demonstrate using stored procedure with ifx\_\* is the car promotion function in our application. The sample code in Example 5-73 includes:

- ▶ **HTML:** The HTML provides a simple form for inserting the data for new promotions.
- ▶ **PHP script:** It does a simple validation for the data sent from the HTML form in the `_POST` array. It either will call the procedure inserting the data or request the promotion data and display it in a very raw format.
- ▶ **Stored procedures:** The `insert_promotion` simply receives the promotion information as the input and inserts the promotion item into the database table. No data is returned to the calling PHP program. `get_promotion_by_car` and `get_car_by_id` will return the promotion information for a specified car if the car is found in the database.

*Example 5-73 Stored procedures and ifx\_\* interface*

---

### 1. HTML

```
<?php
printf("<form method=\"post\" action=\"insert_prom.php\" >");
printf("CarId <input name=\"car\" /><br />");
printf("Amount <input name=\"amount\" /><br />");
printf("Start <input name=\"start\" /><br />");
printf("End <input name=\"end\" /><br />");
printf("<input type=\"submit\" value=\"Confirm\" name=\"Button\"/>");
```



```

        printf("<input type=\"submit\" value=\"List\" name=\"List\"/>");
        printf("<input type=\"submit\" value=\"Abort\" name=\"Cancel\"/>");
        printf("</form>");
?>

```

## 2. PHP script - insert\_prom.php

```

<?php

$DEBUG=1;

$link = @ifx_connect("vps@on1000UC3soc","informix","123456");
if ( !$link ) {
    printf("<pre>");
    printf("Connect: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
    printf("</pre>");
    exit();
}

if (isset($_POST["Button"])) {

    if (!strlen($_POST["car"]) || !strlen($_POST["amount"]) ||
    !strlen($_POST["end"]) || !strlen($_POST["start"]) )
    {
        printf(" Each Field has to be filled out \n");
        exit(1);
    }

    $spstatement="EXECUTE PROCEDURE insert_promotion ( 0, ".
    $_POST["car"] . ", "
    . $_POST["amount"] . ", "
    . $_POST["start"] . "\n", "
    . $_POST["end"] . "\n",
    DEBUG);";

    if ( ! @ifx_query($spstatement,$link) ) {
        if ( !strstr(ifx_error(),"01 I04")) {

            printf("insert_promotion: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
            exit(1);
        }
    }
    printf("insert_promotion: Returncode %s -- %s %s\n",ifx_error(),
    ifx_errormsg(),$spstatement);

}
else if (isset($_POST["List"])) {
/*

```

```

List available promotions for the actual listed car
*/
$spstatement="EXECUTE PROCEDURE get_car_by_id ( ". $_POST["car"] . " );";
if ( ! $result=@ifx_query($spstatement,$link)) ) {
printf("get_car_by_id: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
exit(1);
}
printf("<pre>");

$firstprom=1;
$row=ifx_fetch_row($result);
while ( $row )
{
    print_r($row);

    /* Any Promotion ? */
    $promotion=
"execute procedure get_promotion_by_car( " . $_POST["car"] . " );";
    if ( ! $prom=@ifx_query($promotion,$link)) ) {
printf("PROMO: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
    }
    $prow=ifx_fetch_row($prom);
    while ( $prow )
    {
        if ($firstprom) {
            printf("avaiable promotions \n");
            $firstprom=0;
        }
        print_r($prow);
        $prow=ifx_fetch_row($prom);
    }

    $row=ifx_fetch_row($result);
    $firstprom=1;
}
printf("</pre>");
}

else {
    printf("Promotion module was finished \n");
}
?>

```

### 3. SQL statements for the stored procedures - insert\_promotion, get\_promotion\_by\_car, and get\_car\_by\_id

```

DROP PROCEDURE insert_promotion;
CREATE PROCEDURE insert_promotion (

```

```

p_id integer,
f_m_id integer,
p_amount integer,
p_start varchar(10),
p_end varchar(10),
DEBUG
)
IF DEBUG = 1 then
SET DEBUG FILE to "/tmp/insert_promotion";
TRACE ON;
END IF

INSERT INTO promotion VALUES ( p_id, f_m_id, p_amount, p_start, p_end);
END PROCEDURE;

-- for a spcified car id, all actual valid promotions will returned
-- return clause with resume , more than one promotion could be there

DROP PROCEDURE get_promotion_by_car;
CREATE PROCEDURE get_promotion_by_car (
car_id integer
)
RETURNING integer,integer,integer,varchar(10),varchar(10);

DEFINE local_car_id integer;
DEFINE local_p_id integer;
DEFINE local_p_amount integer;
DEFINE local_p_start char(10);
DEFINE local_p_end char(10);

--SET DEBUG FILE to "/tmp/get_promotion_by_car";
--TRACE ON;
FOREACH c1 for

-- use the input paramter for the SP in the where clause
-- use local variables for the into clause
-- return the values with resume for fetching also the other rows

SELECT *
INTO local_car_id , local_p_id , local_p_amount , local_p_start, local_p_end
FROM promotion
WHERE
    f_m_id = car_id and
    p_start_date < today and
    p_end_date > today

RETURN

```

```

local_car_id , local_p_id , local_p_amount , local_p_start, local_p_end
WITH RESUME;

END FOREACH;
END PROCEDURE;

-- return for a specified car id the complete row from the car table
-- sp is defined with resume, should only return one row per default(PK search)

DROP PROCEDURE get_car_by_id;
CREATE PROCEDURE get_car_by_id (
car_id integer
) RETURNING integer, varchar(30), varchar(30),integer,varchar(30),varchar(30),
varchar(30), varchar(20),varchar(20), integer, decimal(12,2);

DEFINE llocal_m_id integer;
DEFINE llocal_m_type char(50);
DEFINE llocal_m_model char(50);
DEFINE llocal_m_year integer;
DEFINE llocal_m_color char(50);
DEFINE llocal_m_package char(50);
DEFINE llocal_m_ac char(1);
DEFINE llocal_m_power char(1);
DEFINE llocal_m_automatic char(1);
DEFINE llocal_m_doors integer;
DEFINE llocal_m_price decimal(12,2);
DEFINE llocal_m_data char(30000);

--SET DEBUG FILE to "/tmp/get_car_by_id";
--TRACE ON;

FOREACH c1 for

SELECT *
INTO llocal_m_id , llocal_m_type , llocal_m_model , llocal_m_year,
llocal_m_color ,
llocal_m_package , llocal_m_ac , llocal_m_power , llocal_m_automatic ,
llocal_m_doors , llocal_m_price , llocal_m_data
FROM car
WHERE t_id = car_id

RETURN llocal_m_id,
llocal_m_type , llocal_m_model , llocal_m_year, llocal_m_color ,
llocal_m_package , llocal_m_ac , llocal_m_power , llocal_m_automatic ,
llocal_m_doors , llocal_m_price WITH RESUME;

END FOREACH;

```

## Stored procedures with the PEAR DB

We use the inventory module for the car dealer to demonstrate how to work on stored procedure with PEAR DB. Example 5-74 includes:

- ▶ HTML form: For adding, updating, and listing car inventory.
- ▶ PHP script: Depending on the activity initiated by the HTML form, we will either simply validate the input data and call the procedure for inserting the data, or based on who the dealer is of the current user, collect the inventory list from the database and print it in a raw format.
- ▶ Stored procedures: The insert\_car stored procedure will add a car into the database if it is a new car. If the car is already in the database, the car information in the inventory table will be updated. This stored procedure does not return data. The get\_car\_by\_dealer stored procedure lists the car inventory for a specific car dealer.

*Example 5-74 Stored procedures with the DB package*

### 1. HTML

```
<?php
    printf("<form method=\"post\" action=\"maintain_inventory.php\" >");
    printf("type <input name=\"type\" /><br />");
    printf("model <input name=\"model\" /><br />");
    printf("year <input name=\"year\" /><br />");
    printf("color <input name=\"color\" /><br />");
    printf("package <input name=\"package\" /><br />");
    printf("AC <input name=\"ac\" /><br />");
    printf("Windows <input name=\"powerwindows\" /><br />");
    printf("Automatic <input name=\"automatic\" /><br />");
    printf("Doors <input name=\"doors\" /><br />");
    printf("Price <input name=\"price\" /><br />");
    printf("Extras <input name=\"extra\" /><br />");
    printf("<input type=\"submit\" value=\"Add\" name=\"Button\"/>");
    printf("<input type=\"submit\" value=\"Inventory\" name=\"Inv\"/>");
    printf("<input type=\"submit\" value=\"Abort\" name=\"Cancel\"/>");
    printf("</form>");
?>
```

### 2. PHP script - maintain\_inventory.php

```
<?php

$DEBUG=1;
$DEALER=1;
```

```

ini_set("include_path","/usr/local/lib/php");
require_once("DB.php");

$dbh = DB::connect("ifx://informix:123456@on1000UC3soc/vps");

if (isset($_POST["Button"])) {

    if (!strlen($_POST["type"]) || !strlen($_POST["model"]) ||
        !strlen($_POST["year"])
        || !strlen($_POST["color"])
        || !strlen($_POST["package"])
        || !strlen($_POST["ac"])
        || !strlen($_POST["powerwindows"])
        || !strlen($_POST["automatic"])
        || !strlen($_POST["doors"])
        || !strlen($_POST["price"])
        || !strlen($_POST["extra"])
    )
    {
        printf(" Each Field has to be filled out \n");
        exit(1);
    }

    $data=array( $_POST["type"],$_POST["model"], $_POST["year"]
        ,$_POST["color"], $_POST["package"], $_POST["ac"] ,$_POST["powerwindows"],
        $_POST["automatic"], $_POST["doors"] ,$_POST["price"],
        $DEALER, $DEBUG) ;

    $spstatement="EXECUTE PROCEDURE  insert_car ( ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
    ?);";
    $stmt = $dbh->prepare($spstatement);
    if (DB::isError($stmt)) {
        die($stmt->getMessage());
    }
    $dbh->execute($stmt,$data);
    if (DB::isError($dbh)) {
        die($dbh->getMessage());
    }
}
else {
    if (isset($_POST["Inv"])) {

        /*
        dynamic excuting SP
        */

        $data=array($DEALER);
        $spstatement="EXECUTE PROCEDURE  get_car_by_dealer ( ? );";
    }
}

```

```

$stmt = $dbh->prepare($spstatement);
if (DB::isError($stmt)) {
    die($stmt->getMessage());
}
$result = $dbh->execute($stmt,$data);
printf(" %d ",$result);
if (DB::isError($result)) {
    die($result->getMessage());
}
while (($cars = $result->fetchRow())) {
    print_r($cars);
}

/*
calling same SP statically
actually in comment

$spstatement="EXECUTE PROCEDURE get_car_by_dealer ( 1 );";
$result=$dbh->query($spstatement);
if (DB::isError($result)) {
    die($result->getMessage());
}
while (($cars = $result->fetchRow())) {
    print_r($cars);
}

*/

}
else {
    printf("Activity was finished \n");
}
}
?>

```

### 3. SQL Statements for the stored procedures - insert\_car, get\_car\_by\_dealer

```

-- SP for inserting the cars, validation if car already exists
-- no return values

```

```

DROP PROCEDURE insert_car;
CREATE PROCEDURE insert_car (
    m_type varchar(50),
    m_model varchar(50),
    m_year varchar(4),
    m_color varchar(50),
    m_package varchar(50),
    m_ac varchar(1),
    m_power varchar(1),

```

```

m_automatic varchar(1),
m_doors integer,
m_price decimal(12,2),
dealer integer,
DEBUG integer
)

DEFINE local_m_id integer;
DEFINE local_m_type char(50);
DEFINE local_m_model char(50);
DEFINE local_m_year integer;
DEFINE local_m_color char(50);
DEFINE local_m_package char(50);
DEFINE local_m_ac char(1);
DEFINE local_m_power char(1);
DEFINE local_m_automatic char(1);
DEFINE local_m_doors integer;
DEFINE local_m_price decimal(12,2);
DEFINE count integer;

-- use a DEBUG paramter for enabling the trace and the sqexplain
IF DEBUG = 1 then
SET DEBUG FILE to "/tmp/insert_cars";
TRACE ON;
SET EXPLAIN ON;
END IF

-- the car table defines the columns as varchars , use the trim for the input
-- parameter to save space in the database server

LET local_m_type = trim(m_type) ;
LET local_m_model = trim(m_model);
LET local_m_year = m_year ;
LET local_m_color = trim(m_color) ;
LET local_m_package = trim(m_package) ;
LET local_m_ac = trim(m_ac) ;
LET local_m_power = trim(m_power) ;
LET local_m_automatic = trim(m_automatic);

-- does the already exists ?

SELECT t_id,count(*) INTO local_m_id,count
FROM car
WHERE
    t_type = local_m_type and
    t_model = local_m_model and
    t_year::int = local_m_year and
    t_color = local_m_color and
    t_package = local_m_package and

```



```

        t_ac = local_m_ac and
        t_powerwindows = local_m_power and
        t_automatic = local_m_automatic and
        t_doors = m_doors and
        t_price = m_price
    GROUP BY 1;

    if count > 0
    then
    UPDATE inventory SET n_quantity = n_quantity+1
    WHERE f_d_id=dealer AND f_m_id =local_m_id;
    else
    INSERT INTO car VALUES (
    0, m_type , m_model , m_year , m_color , m_package , m_ac ,
    m_power , m_automatic , m_doors , m_price ,"" );

    -- use dbinfo for determine the serial value for the insert

    LET local_m_id=dbinfo("sqlca.sqlerrd1");
    INSERT INTO inventory VALUES ( dealer,local_m_id,1 );
    end if

END PROCEDURE;

-- return all cars for a specific dealer id
-- use resume in the return clause to allow more than one row returned

DROP PROCEDURE get_car_by_dealer;
CREATE PROCEDURE get_car_by_dealer (
dealer integer
) RETURNING integer, varchar(30), varchar(30),integer,varchar(30),varchar(30),
varchar(30), varchar(20),varchar(20), integer, decimal(12,2),integer;

DEFINE llocal_m_id integer;
DEFINE llocal_m_type char(50);
DEFINE llocal_m_model char(50);
DEFINE llocal_m_year integer;
DEFINE llocal_m_color char(50);
DEFINE llocal_m_package char(50);
DEFINE llocal_m_ac char(1);
DEFINE llocal_m_power char(1);
DEFINE llocal_m_automatic char(1);
DEFINE llocal_m_doors integer;
DEFINE llocal_m_price decimal(12,2);
DEFINE llocal_m_data char(30000);
DEFINE llocal_n_quantity integer;

--SET DEBUG FILE to "/tmp/get_car_by_dealer";
--TRACE ON;

```

```

-- use the foreach for the cursor
-- addition return the value with resume to get more rows with the next fetch

FOREACH c1 for

SELECT car.*,n_quantity
INTO llocal_m_id , llocal_m_type , llocal_m_model , llocal_m_year,
llocal_m_color ,
llocal_m_package , llocal_m_ac , llocal_m_power , llocal_m_automatic ,
llocal_m_doors , llocal_m_price , llocal_m_data , llocal_n_quantity
FROM car,inventory
WHERE t_id = f_m_id and f_d_id= dealer

RETURN llocal_m_id,
llocal_m_type , llocal_m_model , llocal_m_year, llocal_m_color ,
llocal_m_package , llocal_m_ac , llocal_m_power , llocal_m_automatic ,
llocal_m_doors , llocal_m_price , llocal_n_quantity WITH RESUME;

END FOREACH;
END PROCEDURE;

```

---

### **Note**

The PEAR DB package in Version 1.7.6 does not support stored procedures returning result sets. The stored procedure will be executed but you will not be able to access the result set. In this case, a small change in the provided PHP files will provide the support. For the changes, do the following steps:

1. Identify the directory for the `ifx.php` script.
2. Search the function `simpleQuery` in the file.
3. Change the code and add the following lines:

```

// Determine which queries should return data, and which
// should return an error code only.
if (preg_match('/(EXECUTE PROCEDURE)/i', $query)) { <-- add here
    return $result;                                <-- add here
}                                                    <-- add here
if (preg_match('/(SELECT)/i', $query)) {
    return $result;
}

```

### **Stored procedures with unixODBC**

We use Example 5-75 for the discussion of handling stored procedures with the unixODBC extension for the order process. The example covers the order of a

new or used car for an user and provides the ability to see actual car reviews for a given car. It contains:

- ▶ **HTML:** For adding the car details and initiating the order process. The reviews can be retrieved with the same form.
- ▶ **PHP script:** Once the data is typed, the script validates the data and calls the appropriate stored procedure. The data for the review will be returned in a raw format on the panel.
- ▶ **Stored Procedures:** In the SQL statements for the procedures, we define `insert_order` for inserting the submitted order in the database table. This stored procedure will not return a value. The procedure `get_review_bycar` will select the car based on the input parameter and return the car details and the existing review if any exists.

#### *Example 5-75 Using stored procedures with static SQL in unixODBC*

---

##### **1. HTML form**

```
<?php
    printf("<form method=\"post\" action=\"maintain_orders.php\" >");
    printf("type <input name=\"type\" /><br />");
    printf("model <input name=\"model\" /><br />");
    printf("year <input name=\"year\" /><br />");
    printf("color <input name=\"color\" /><br />");
    printf("package <input name=\"package\" /><br />");
    printf("AC <input name=\"ac\" /><br />");
    printf("Windows <input name=\"powerwindows\" /><br />");
    printf("Automatic <input name=\"automatic\" /><br />");
    printf("Doors <input name=\"doors\" /><br />");
    printf("Price <input name=\"price\" /><br />");
    printf("Extras <input name=\"extra\" /><br />");
    printf("<input type=\"submit\" value=\"Buy\" name=\"Button\"/>");
    printf("<input type=\"submit\" value=\"Get Reviews\" name=\"Rev\"/>");
    printf("<input type=\"submit\" value=\"Abort\" name=\"Cancel\"/>");
    printf("</form>");
?>
```

##### **2. PHP Script with unixODBC - maintain\_orders.php**

```
<?php

$DEBUG=1;
$DEALER=1;

$cn =
odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=vps","informix","123
456"
);
```

```

if ( !$cn ) {
    printf("<pre>Connect: attempt to connect was not successful \n</pre>");
    exit();
}

if (!strlen($_POST["type"])
|| !strlen($_POST["model"])
|| !strlen($_POST["year"])
|| !strlen($_POST["color"])
|| !strlen($_POST["package"])
|| !strlen($_POST["ac"])
|| !strlen($_POST["powerwindows"])
|| !strlen($_POST["automatic"])
|| !strlen($_POST["doors"])
|| !strlen($_POST["price"])
|| !strlen($_POST["extra"])
)
{
    printf(" Each Field has to be filled out \n");
    exit(1);
}

$spstatement="EXECUTE PROCEDURE insert_order(" .
"\\"" . $_POST["type"] . "\"\"," .
"\\"" . $_POST["model"] . "\"\"," .
"\\"" . $_POST["year"] . "\"\"," .
"\\"" . $_POST["color"] . "\"\"," .
"\\"" . $_POST["package"] . "\"\"," .
"\\"" . $_POST["ac"] . "\"\"," .
"\\"" . $_POST["powerwindows"] . "\"\"," .
"\\"" . $_POST["automatic"] . "\"\"," .
$_POST["doors"] . "," .
$_POST["price"] . "," .
. $DEALER . "," .
. $DEBUG . " );";

if (isset($_POST["Button"])) {
    $stmt = odbc_prepare ($cn, $spstatement);
    if ( !$stmt ) {
        printf("Pinsert_order %s %s \n",odbc_error($cn), odbc_errormsg($cn));
        exit();
    }
    $odbc_result = odbc_execute($stmt);
}
else {

```

```

if (isset($_POST["Rev"])) {

$spstatement="EXECUTE PROCEDURE  get_review_bycar(" .
"\\"" . $_POST["type"] . "\"\"," .
"\\"" . $_POST["model"] . "\"\"," .
"\\"" . $_POST["year"] . "\"\"," .
"\\"" . $_POST["color"] . "\"\"," .
"\\"" . $_POST["package"] . "\"\"," .
"\\"" . $_POST["ac"] . "\"\"," .
"\\"" . $_POST["powerwindows"] . "\"\"," .
"\\"" . $_POST["automatic"] . "\"\"," .
$_POST["doors"] . "," . $DEBUG . ");";

/* will not work !
$stmt = odbc_prepare ($cn, $spstatement);
if ( !$stmt ) {
    printf("Pget_review_ %s %s \n",odbc_error($cn), odbc_errormsg($cn));
    exit();
}
*/
$resultset = odbc_exec($cn,$spstatement);
if ( !$resultset ) {
    printf("Pget_review_ %s %s \n",odbc_error($cn), odbc_errormsg($cn));
    exit();
}

printf("<pre>Available reviews for the selected car:\n\n");
while ( @($preview = odbc_fetch_array($resultset)) ) {
    print_r($preview);
}
printf("</pre>");

}
else {
    printf("New Order module was finished \n");
}
}
?>

```

### 3. SQL statements for the stored procedures in the database server - insert\_order and get\_review\_bycar

```

dbaccess << EOF
database vps;

CREATE TABLE review (
r_id integer,
f_m_id integer ,

```

```

f_c_id integer,
r_date date,
r_data lvarchar(500)); -- use a lvarchar here to allow larger reviews

-- a new customer order will be inserted
-- inventory table will be review that the is available
-- no value will be returned or a custom error -746, errorhandling in PHP

DROP PROCEDURE insert_order;
CREATE PROCEDURE insert_order (
m_type varchar(50),
m_model varchar(50),
m_year  varchar(4),
m_color varchar(50),
m_package varchar(50),
m_ac varchar(1),
m_power varchar(1),
m_automatic varchar(1),
m_doors integer,
m_price decimal(12,2),
dealer integer,
DEBUG integer
)

DEFINE local_m_id integer;
DEFINE local_m_type char(50);
DEFINE local_m_model char(50);
DEFINE local_m_year integer;
DEFINE local_m_color char(50);
DEFINE local_m_package char(50);
DEFINE local_m_ac char(1);
DEFINE local_m_power char(1);
DEFINE local_m_automatic char(1);
DEFINE local_m_doors integer;
DEFINE local_m_price decimal(12,2);
DEFINE count integer;
DEFINE local_f_d_id integer;

IF DEBUG = 1 THEN
SET DEBUG FILE to "/tmp/insert_order";
TRACE ON;
SET EXPLAIN ON;
END IF

let local_m_type = trim(m_type) ;
let local_m_model = trim(m_model);
let local_m_year = m_year ;
let local_m_color = trim(m_color) ;

```

```

let local_m_package = trim(m_package) ;
let local_m_ac = trim(m_ac) ;
let local_m_power = trim(m_power) ;
let local_m_automatic = trim(m_automatic);

SELECT  t_id,f_d_id,count(*) INTO local_m_id,local_f_d_id,count
FROM car,inventory
WHERE
    t_type = local_m_type and
    t_model = local_m_model and
    t_year::int = local_m_year and
    t_color = local_m_color and
    t_package = local_m_package and
    t_ac = local_m_ac and
    t_powerwindows = local_m_power and
    t_automatic = local_m_automatic and
    t_doors = m_doors and
    t_price = m_price and
    t_id=f_m_id and
    f_d_id=dealer and
    n_quantity>0
GROUP BY 1,2;

IF count > 0
THEN
UPDATE inventory set n_quantity = n_quantity-1
WHERE f_d_id=dealer and f_m_id =local_m_id;
INSERT INTO orders VALUES ( 0,local_m_id,DEALER,today,today+14);
else
RAISE EXCEPTION -746 , 0, "Car not found or not available ";
END IF

END PROCEDURE;

-- input paramter are a car attributes
-- try to get the car_id for that attributes
-- determine all reviews for that id
-- return the values with resume to anble multiple rows returned

DROP PROCEDURE get_review_bycar;
CREATE PROCEDURE get_review_bycar (
m_type varchar(50),
m_model varchar(50),
m_year varchar(4),
m_color varchar(50),
m_package varchar(50),
m_ac varchar(1),
m_power varchar(1),

```

```

m_automatic varchar(1),
m_doors integer,
DEBUG integer
)
RETURNING integer, char(50), char(50), lvarchar(500);

```

```

-- for the car
DEFINE local_m_id integer;
DEFINE local_m_type char(50);
DEFINE local_m_model char(50);
DEFINE local_m_year integer;
DEFINE local_m_color char(50);
DEFINE local_m_package char(50);
DEFINE local_m_ac char(1);
DEFINE local_m_power char(1);
DEFINE local_m_automatic char(1);
DEFINE local_m_doors integer;

```

```

--for the review
DEFINE local_r_id integer;
DEFINE local_r_m_id char(50);
DEFINE local_r_date char(50);
DEFINE local_r_data lvarchar(500);

```

```

IF DEBUG = 1 then
SET DEBUG FILE to "/tmp/get_review_bycar";
TRACE ON;
SET EXPLAIN ON;
END IF

```

```

LET local_m_type = trim(m_type) ;
LET local_m_model = trim(m_model);
LET local_m_year = m_year ;
LET local_m_color = trim(m_color) ;
LET local_m_package = trim(m_package) ;
LET local_m_ac = trim(m_ac) ;
LET local_m_power = trim(m_power) ;
LET local_m_automatic = trim(m_automatic);

```

```

FOREACH c1 FOR

```

```

SELECT t_id INTO local_m_id
FROM car
WHERE
    t_type = local_m_type and
    t_model = local_m_model and

```



```

t_year::int = local_m_year and
t_color = local_m_color and
t_package = local_m_package and
t_ac = local_m_ac and
t_powerwindows = local_m_power and
t_automatic = local_m_automatic and
t_doors = m_doors

FOREACH c2 FOR
SELECT r_id,f_m_id,r_date,r_data
INTO local_r_id , local_r_m_id , local_r_date , local_r_data
FROM review WHERE f_m_id=local_m_id

RETURN local_r_id , local_r_m_id , local_r_date , local_r_data WITH RESUME;

END FOREACH;
END FOREACH;
END PROCEDURE;

```

---

In the unixODBC extension, you can only use the `odbc_exec` function for executing stored procedures returning result sets. This means that the call of the stored procedure must be static. If using dynamic SQL for executing stored procedures with `odbc_prepare` and `odbc_execute`, the procedure will be executed at the database server but will not return any values. This limitation is similar to the usage of the `SELECT` statement, but does not exist for procedures which do not return any value.

The ODBC library provided by the Informix SDK 2.90 or Informix Connect 2.90 has a restriction using dynamic SQL statements executing a stored procedure returning no values. Using specific data types as char-based data types the ODBC driver will truncate the value of the variable to "" and return a warning, "S01004 Data truncated". The procedure will be executed, but the result will be undefined, depending on the logic. Use the static SQL as a workaround and avoid the prepare.

## 5.2.6 BLOB and SBLOB data types

In this section, we discuss how to work with BLOB and SBLOB data types in Informix IDS with the various PHP database extensions.

### BLOB data type

With BLOBs, Informix IDS can handle large objects as binary data in a byte data type and text objects in a text data type. The BLOB data type provides you the capability to store images and entire documents in the database. These data types have been supported by Informix IDS since Version 4. BLOB data types

can be stored within all the other data in the table space, or in a separate specified blobspace.

BLOB data types can be used in several operational areas. Commonly document retrieval systems and geographic information systems are based on this data type. The retrieval of BLOB data with SQL needs the define of keywords, which are stored together with the BLOB in the data row. The benefit of storing large data in the database is, beside an easy search via keywords and a combination of data stored in different rows but belonging together, also the opportunity of backup and restore and the apply of delete and update for specific data. Storing the data in a file in the operating system would cause much more maintenance effort.

In our Dealership example, we use the BLOBs to store car pictures and customer reviews. In this section, we use the following table to discuss how to work with BLOBs with various Informix database interfaces.

```
CREATE TABLE large ( key SERIAL, photo BYTE, review TEXT);
```

The photo column can be taken as an example for a car photo in our car dealer application and review as a review of a purchased car added by the buyer.

### ***Informix PDO and BLOB data***

Example 5-76 shows, with Informix PDO, how to insert the large data from a file into a table and select the same data back into a file in the local file system. Be aware that the resulting data of the large objects is in data streams so you need to use the *fread* function to read the data stream.

#### ***Example 5-76 Select and insert BLOB data with Informix PDO***

---

##### **PHP Script:**

```
<?php

$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");
if (!$dbh) { exit(); }

/*
try to insert the BLOB
*/
$stmt= $dbh->prepare("INSERT INTO large VALUES (0,?,?)");

$file = fopen ("/tmp/review","r");
$image1 = fread ( $file, 100000000) ;
fclose ( $file);
$file = fopen ("/tmp/photo","r");
$image2 = fread ( $file, 100000000) ;
```

```

fclose ( $file);

$stmt->bindParam(1, $image1 );
$stmt->bindParam(2, $image2);

$stmt->execute();
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" execute insert blobs failed with %s \n",$error["1"]);
    exit(1);
}

/*
    try to get the BLOB from the database back
*/

$query=$dbh->query("SELECT * FROM large ");
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" select blobs failed with %s \n",$error["1"]);
    exit(1);
}

$count=1;
$row=$query->fetch(PDO::FETCH_ASSOC);
while ( $row ) {
    $file = fopen ("/tmp/large.photo.$count","w");
    $test=fread($row["PHOTO"],20000);
    while($test) {
        fwrite ( $file, $test) ;
        $test=fread($row["PHOTO"],20000);
    }
    fclose ( $file);
    $file = fopen ("/tmp/large.descr.$count","w");
    $test1=fread($row["DESCR"],20000);
    while($test1) {
        fwrite ( $file, $test1) ;
        $test1=fread($row["DESCR"],20000);
    }
    fclose ( $file);

    printf(" %d \n",$row["CAR_ID"]);
    $count++;
    $row=$query->fetch(PDO::FETCH_ASSOC);
}

?>

```

**Result:**

```
ls -al /tmp/large*
-rw-r--r-- 1 root root 30 Feb 14 09:18 /tmp/large.descr.1
-rw-r--r-- 1 root root 30 Feb 14 09:18 /tmp/large.descr.2
-rw-r--r-- 1 root root 30 Feb 14 09:18 /tmp/large.descr.3
-rw-r--r-- 1 root root 2861 Feb 14 09:18 /tmp/large.descr.4
-rw-r--r-- 1 root root 30 Feb 14 09:18 /tmp/large.photo.1
-rw-r--r-- 1 root root 30 Feb 14 09:18 /tmp/large.photo.2
-rw-r--r-- 1 root root 30 Feb 14 09:18 /tmp/large.photo.3
-rw-r--r-- 1 root root 2861 Feb 14 09:18 /tmp/large.photo.4
```

---

The BLOB data to be inserted can be read from the file without placing it in a variable first. This simplifies the code somewhat. See Example 5-77.

*Example 5-77 Insert BLOBs directly from a file*

---

```
<?php

$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");
if (!$dbh) { exit(); }

/*
try to insert the BLOB
*/

$stmt= $dbh->prepare("INSERT INTO large VALUES (0,?,?)");
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" prepare insert failed with %s \n",$error["1"]);
    exit(1);
}

$file = fopen ("/tmp/large.photo","r");
$file1 = fopen ("/tmp/large.descr","r");

$stmt->bindParam(1, $file, PDO::PARAM_LOB);
$stmt->bindParam(2, $file1, PDO::PARAM_LOB);

$stmt->execute();
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" execute insert blobs failed with %s \n",$error["1"]);
    exit(1);
}
?>
```

---

In Example 5-76 on page 304, the data was selected as a stream in string data type. Example 5-78 shows how to bind variables to a SELECT. The data will be placed in the variables by column.

---

*Example 5-78 Select BLOBs into bound parameters with Informix PDO*

---

```
<?php
$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");
if (!$dbh) { exit(); }

/*
try to select the BLOB from the database into bind strings
*/

$stmt=$dbh->query("SELECT * FROM large ");

$count=1;
$str="";
$str1="";
$id=0;
$stmt->bindColumn(1, $id, PDO::PARAM_INT);
$stmt->bindColumn(2, $str, PDO::PARAM_STR,20000);
$stmt->bindColumn(3, $str1, PDO::PARAM_STR,20000);

while ( $stmt->fetch(PDO::FETCH_BOUND) )
{

    $file = fopen ("/tmp/large.photo.$count","w");
    $file1 = fopen ("/tmp/large.descr.$count","w");
    print_r($id);
    fwrite($file,$str);
    fwrite($file1,$str1);
    fclose ($file);
    fclose ($file1);
    $count++;

}
?>
```

---

Example 5-79 shows how to update existing BLOB fields with Informix PDO.

---

*Example 5-79 Update BLOBs with Informix PDO*

---

```
<?php
$dbh = new PDO("informix:host=polonium; service=22222; database=vps;
server=on1000UC3soc; protocol=onsoctcp ", "informix", "123456");
if (!$dbh) { exit(); }
```

```

/*
try to update the BLOB columns
*/
$stmt= $dbh->prepare("UPDATE large SET descr=? , photo=? ");
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" prepare update blob columns failed with %s \n",$error["1"]);
    exit(1);
}

$descr="This is an PDO descr blob text";
$photo="This is an PDO photo blob text";
$stmt->bindParam(1, $descr);
$stmt->bindParam(2, $photo);

$stmt->execute();
$error=$dbh->errorInfo();

?>

```

---

### ***ifx\_\* functions and BLOB data***

Since *ifx\_\** is based on Informix ESQL/C, it uses the same memory structure defined in ESQL/C to handle the BLOBs. The major structure handling BLOBs in ESQL/C is a locator structure. As the name implies, on all the attributes of locator structure, the main one is to define where the content of the BLOB resides.

There are two ways to store BLOBs: One stores the BLOB described in the locator structure in a file, and the other one stores the BLOB in memory. The *ifx.blobinfile* in the *php.ini* is the global setting parameter for the BLOB storage location. Set *ifx.blobinfile* = 0 will get the BLOB in memory; set the value to 1 to store the BLOB in a file. You can also change the location of the BLOB dynamically in a PHP program to overwrite the global settings from the *php.ini* file. Use the function *ifx\_blobinfile\_mode()*. The meanings of the parameter values are similar to the global settings.

Another mechanism has to be described, too. Definitions of *ifx.textasvarchar* and *ifx.byteasvarchar* in the *php.ini* define how the PHP extension will return the specific BLOB to the program. If you set this to 1, the select will return the BLOB as a varchar in the row. If you set this to 0, the select will return an internal BLOB structure and the row will only contain a reference in kind of a value for the BLOB column. Using the reference after the fetch, the content has to be managed with the *ifx\_get\_blob* function. Both parameters can be set independently. Use the functions *ifx\_byteasvarchar* and *ifx\_textasvarchar* to define the setting in the program.

Example 5-80 on page 309 shows how to insert BLOB data in PHP with `ifx_*`. The example includes both inserting BLOB data from a file and variables. Note that you can only use dynamic SQL for BLOBs.

*Example 5-80 Insert BLOB with ifx\_\* functions*

---

```
<?php
$link = ifx_connect("vps@on1000UC3soc","informix","123456");

/*
from a file
*/

$text = ifx_create_blob(0, 1, "/tmp/review");
$byte = ifx_create_blob(0, 1, "/tmp/photo");
$blobidarray[] = $text;
$blobidarray[] = $byte;

if (!@ifx_query("INSERT INTO large VALUES (0 , ? , ?)", $link, $blobidarray)) {
    printf("<pre>");
    printf("Connect: Returncode %s -- %s \n", ifx_error(), ifx_errormsg());
    printf("</pre>");
    exit();
}

/*
from memory
*/
$file=fopen("/tmp/review","r");
$vtext=fread($file,100000);
fclose($file);

$file=fopen("/tmp/photo","r");
$vbyte=fread($file,100000);
fclose($file);

$text = ifx_create_blob(0, 0, $vtext);
$byte = ifx_create_blob(0, 0, $vbyte);
$blobidarray[] = $text;
$blobidarray[] = $byte;

if (!($stmt=@ifx_prepare("INSERT INTO large VALUES (0 , ? , ?)", $link,
                        $blobidarray))) {
    printf("<pre>");
    printf("Connect: Returncode %s -- %s \n", ifx_error(), ifx_errormsg());
    printf("</pre>");
    exit();
}
ifx_do($stmt);
```

?>

Example 5-81 shows how to get the BLOB data from the database. The BLOB data can be selected into a file or variables. These two ways of handling BLOB data are shown in the body of the WHILE clause with the use of `ifx_fetch_row()`.

---

*Example 5-81 Select BLOB data with ifx\_\* functions*

---

**1. select into a FILE**

```
<?php

$link = ifx_connect("vps@on1000UC3soc","informix","123456");
/*
select the BLOB into a FILE,
*/

ifx_blobinfile_mode(1);
ifx_byteasvarchar(0);
ifx_textasvarchar(0);

if (! ($result=@ifx_query("SELECT * FROM large ", $link))) {
    printf("<pre>");
    printf("Connect: Returncode %s -- %s \n", ifx_error(), ifx_errormsg());
    printf("</pre>");
    exit();
}

while ( $row=ifx_fetch_row($result))
{
    print_r($row);

    $fp=fopen(ifx_get_blob($row["photo"]), "r");
    $test=fread($fp, 100000);
    fclose($fp);
    printf(" %-20s ", $test);

    $fp=fopen(ifx_get_blob($row["descr"]), "r");
    $test=fread($fp, 100000);
    fclose($fp);
    printf(" %-20s ", $test);

}
?>
```

**2. SELECT into Memory**

```
<?php

$link = ifx_connect("vps@on1000UC3soc","informix","123456");
```



```

/*
representation as blob object
*/
ifx_blobinfile_mode(0);
ifx_byteasvarchar(0);
ifx_textasvarchar(0);

if (! ($result=@ifx_query("SELECT * FROM large ",$link))) {
    printf("<pre>");
    printf("Connect: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
    printf("</pre>");
    exit();
}

while ( $row=ifx_fetch_row($result))
{
    printf(" %s %s ",ifx_get_blob($row["descr"]),ifx_get_blob($row["photo"]));
}
/*
representation as varchar
*/
ifx_blobinfile_mode(0);
ifx_byteasvarchar(1);
ifx_textasvarchar(1);

if (! ($result=@ifx_query("SELECT * FROM large ",$link))) {
    printf("<pre>");
    printf("Connect: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
    printf("</pre>");
    exit();
}

while ( $row=ifx_fetch_row($result))
{
    printf(" %s %s ",$row["descr"],$row["photo"]);
}
?>

```

---

To close our discussion about BLOBs in ifx\_\*, we have a small example (Example 5-82) that shows how to update a BLOB column. This example creates two BLOB objects from strings in memory and updates the database with a dynamic SQL. You can easily change the script to create a BLOB object from a file source as shown in Example 5-80 on page 309.

---

*Example 5-82 Update BLOB columns with ifx\_\* functions*

---

```
<?php

$link = ifx_connect("vps@on1000UC3soc","informix","123456");

/*
create a BLOB object from Memory
*/

$textstring="This is an updated ifx text blob ";
$bytestring="This is an updated ifx byte blob ";

$text = ifx_create_blob(0, 0, $textstring);
$byte = ifx_create_blob(0, 0, $bytestring);

$blobidarray[] = $text;
$blobidarray[] = $byte;

if (! @ifx_query("UPDATE large SET descr=?, photo=?", $link, $blobidarray)) {
    printf("<pre>");
    printf("Connect: Returncode %s -- %s \n", ifx_error(), ifx_errormsg());
    printf("</pre>");
    exit();
}
?>
```

---

***PEAR DB package and BLOB data***

PEAR DB Version 1.7.6 does not support BLOB data types. If you try to insert the BLOB as a simple string, this will fail with a conversion error. Similarly, a select from a table with BLOB data also returns an error. This is based on the restriction in the database server that BLOBs are not allowed for use with scroll cursors. But the scroll cursor will be used in PEAR DB as the only possibility to get data. But if there is the specific need for BLOBs in that environment, you could use the ifx\_\* functionality as an exception in parallel.

***unixODBC and BLOB data***

UnixODBC has an automatic data conversion included. This means you can use a simple string variable for the SQL statements for manipulating the BLOB data. It is much simpler to use in comparison with the other interfaces. We discuss how to select, insert, and update BLOB data with unixODBC by examples.

Example 5-83 shows a simple way to insert a string into a BLOB column.

---

*Example 5-83 insert simple BLOB with unixODBC*

---

```
<?php
```

```

$cn =
odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=vps","informix","123
456"
);

/*
insert the BLOB with a dynamic SQL and a string as parameters
*/
$parm1=0;
$parm2="This is an ODBC Text Blob";
$parm3="This is an ODBC Byte Blob";

$str="INSERT INTO large VALUES ( ?,?, ?)";
$stmt = odbc_prepare ($cn, $str);
if ( !$stmt ) {
    printf("PREPARE %s %s \n",odbc_error($cn), odbc_errormsg($cn));
    exit();
}

/*
    compilation of the different variables to an array at exec time
*/

$odbc_result = odbc_execute($stmt,array($parm1,$parm2,$parm3));
if ( !$odbc_result ) {
    printf("PREPARE %s %s \n",odbc_error($cn), odbc_errormsg($cn));
    exit();
}
@odbc_close($cn);
?>

```

---

Example 5-84 shows how to select the BLOB data. Again, there is an automatic conversion of the BLOB into a string.

---

*Example 5-84 Select BLOB data with unixODBC*

---

```

<?php

$cn =
odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=vps","informix","123
456"
);

$str="SELECT * FROM large";

$odbc_result = odbc_exec($cn,$str);
if ( !$odbc_result ) {
    printf(" %s %s \n",odbc_error($cn), odbc_errormsg($cn));
    exit();
}

```

```

}

printf("<pre>");
while ( odbc_fetch_row($odbc_result) )
{
for ( $count=1; $count <= odbc_num_fields($odbc_result); $count ++ ) {
printf(" %s ",odbc_result($odbc_result,$count));
}
printf(" \n#####\n");
}
printf("</pre>");
@odbc_close($cn);
?>

```

---

You can update a BLOB column in a similar way. Example 5-85 shows how to update the BLOB with a string defined in the PHP script. We use dynamic SQL for the update statement.

*Example 5-85 Update a BLOB column with unixODBC*

---

```

<?php

$cn =
odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=vps","informix","123
456"
);

/*
    Using odbc_exec for the update with ? and parameter list
*/
$parm1=0;
$parm2="This is an updated ODBC Text Blob";
$parm3="This is an updated Text Blob";

$str="UPDATE large SET photo=?, descr=? ";
$stmt = odbc_prepare ($cn, $str);
if ( !$stmt ) {
    printf("PREPARE %s %s \n",odbc_error($cn), odbc_errormsg($cn));
    exit();
}

/*
    compilation of the different Variables to an array at exec time
*/

$odbc_result = odbc_execute($stmt,array($parm2,$parm3));
if ( !$odbc_result ) {
    printf("EXECUTE BLOB update %s %s \n",
        odbc_error($cn), odbc_errormsg($cn));
}

```

```

        exit();
    }

    @odbc_close($cn);
?>

```

---

## SBLOBs

Since the particular BLOB data type we have discussed in detail was implemented a long time ago, and expectations for maintaining large data types have increased, the Informix IDS (Versions 9 and 10) provides additional data types to access large data. There are the BLOB and the CLOB data types. We also refer to them as SBLOBs. They are stored in an sblobspace (smart blob space) in the database server. More advanced than the BLOB data types, SBLOBs provide more flexibility for searching in the data (which was impossible with BLOBs), and there is an additional set of functions defined in the server that provides an API for access. DataBlades are a common area for using the SBLOBs. DataBlades are extensions of the default Informix IDS functionality.

We do not discuss the usage of the SBLOBs in detail for the extensions. To use the SBLOBs, perform the following steps in the database server:

1. Create an sblobspace in the database server.
2. Change the SBSPACENAME to the newly created sblobspace.
3. Restart the server with onmode -ky and oninit.
4. Create your table, taking the following DDL as an example:

```
CREATE TABLE large ( key SERIAL, photo BLOB, review CLOB);
```

We have tested the Informix PDO examples from the BLOB discussion (Example 5-76 on page 304 to Example 5-79 on page 307) with a table using SBLOB data types. They can be used without any change in the PHP code.

### 5.2.7 Error handling

Error handling is important for all applications, including Web-based applications. Using our Dealership application as an example, what if there is a database problem and a message such as this one pops up on the panel while the user is using the the application?

```
Warning: odbc_connect(): SQL error: [unixODBC][Informix][Informix ODBC
Driver][Informix]User (informix) password "123456" is not able to connect
for the database server, Server is down, SQL state 28000 in SQLConnect in
/usr/local/apache2/htdocs/odbc/error/error.php on line 4
```

This not only spoils the well-designed Web page, but also creates a security exposure. Following are some basic concepts related to error handling in the PHP with various database interface areas:

- ▶ PDO, `ifx_*`, and `unixODBC` functions do not suppress the error messages by default. Failed functions may report errors to the panel. This applies to all functions including the connectivity functions which are the common source for unexpected errors. To control the errors displayed in the window, you can:
  - Modify the settings `php.ini` file: `php.ini` file contains the parameters which control the level of the messages to be displayed. You can suppress all the messages to be displayed on the window. Changing the settings of `error_reporting` and `display_errors` can be taken as a starting point.
  - The `@` operator in PHP can also suppress messages from the called function appearing in the browser. Use this operator as a prefix before you call, especially functions where the function logic is out of the control of your program. A good example for the usage of this operator is the PEAR DB package. Look in the `ifx.php` file and search for the used `ifx_*` functions.
- ▶ PHP 5 has introduced, as a part of the new object-oriented programming interface, the exceptions which are already used for other programming languages. We strongly suggest you consider exceptions for the usage of Informix PDO. Additionally, you can advance this interface by creating, throwing, and catching your own exceptions.

In addition to database errors, there are other error conditions which could happen in a Web application, such as broken links, missing scripts, and exceptions in CGI executables. These all must be handled by error handling routines. The primary goal is to catch the errors, log them internally, execute appropriate activities like sending an e-mail to the Web master, and display a meaningful message to users.

## Errors and exceptions in Informix PDO

Informix PDO defines, in comparison with the procedural-oriented interface, two ways of expressing an error in a database environment:

- ▶ Exceptions raised internally by the PDO based on an error condition
- ▶ Database-generated errors. This type of exception can be captured and handled by calling the PDO class `errorInfo()` or `errorCode()` function.

Example 5-86 shows two exceptions raised by Informix PDO functions. One is a connection request to the database that failed because the specified database does not exist and the other is an attempt to start a transaction twice.

**Example 1:**

```
<?php

$dbh = new PDO("informix:: database=vps_ansi00; server=on1000UC3soc;",
"informix", "123456");

?>
```

Output:

Fatal error: Uncaught exception 'PDOException' with message 'SQLSTATE=HY000, SQLDriverConnect: -329 [Informix][Informix ODBC Driver][Informix]Database not found or no system permission.' in /usr/local/apache2/htdocs/pdo/error/excpetion1.php:4

Stack trace:

```
#0 /usr/local/apache2/htdocs/pdo/error/excpetion1.php(4):
PDO->__construct('informix:: data...', 'informix', '123456')
#1 {main}
   thrown in /usr/local/apache2/htdocs/pdo/error/excpetion1.php on line 4
```

**Example 2:**

```
<?php

$dbh = new PDO("informix:: database=vps_ansi; server=on1000UC3soc;",
"informix", "123456");

    $dbh->beginTransaction();
    $dbh->beginTransaction();

?>
```

Fatal error: Uncaught exception 'PDOException' with message 'There is already an active transaction' in /usr/local/apache2/htdocs/pdo/error/excpetion.php:7

Stack trace:

```
#0 /usr/local/apache2/htdocs/pdo/error/excpetion.php(7):
PDO->beginTransaction()
#1 {main}
   thrown in /usr/local/apache2/htdocs/pdo/error/excpetion.php on line 7
```

---

If these exceptions are not caught and processed, the application terminates. Example 5-87 shows the usage of the basic exception handler provided by PHP 5 to cover these errors.

The action which should be taken once an exception is caught depends on where the error originates. For example, if the error occurs in the connecting to the database phase and the application cannot continue, the action should be to

generate an “out of order” Web page with contact details. If it is a minor database error, logging the error and retrying the activity should be an appropriate action.

*Example 5-87 Catch exceptions within PDO*

---

```
<?php
try
{
$dbh = new PDO("informix:: database=vps_ansi; server=on1000UC3soc;",
"informix", "123456");

    $dbh->beginTransaction();
    $dbh->beginTransaction();
}
catch (PDOException $e )
{
    printf("Error: %s \n",$e->getMessage());
}
?>
```

**Output:**

Error: There is already an active transaction.

---

The error information generated during executing the SQL statement in the database server is different from the exceptions generated by the Informix PDO extension. The Informix PDO function `errorInfo()` can be used to capture the status of the last executed SQL statement in the database. This function returns an array with three elements, the `sqlstate`, the `sqlcode`, and the error message. The details of the meaning of the codes are described in *ESQL/C Programmer's Manual, Version 9.53*, G251-1342. The function `errorCode()` is available to retrieve the SQL statement status. This function returns only the `sqlstate` information.

Example 5-88 shows how to use the `errorInfo()` and what the output looks like.

*Example 5-88 Database-generated errors in Informix PDO*

---

```
<?php

$dbh = new PDO("informix:: database=vps; server=on1000UC3soc;",
"informix", "123456");

$stmt=$dbh->query('SELECT * FROM nonexistingtable ');
/*
    question the error code
    output of the error Routines
*/
```



```

$error=$dbh->errorInfo();
print_r($error);
if (!$error[1])
$row=$stmt->fetch(PDO::FETCH_ASSOC);

?>

```

**Output:**

```

Array
(
    [0] => 42S02
    [1] => -206
    [2] => [Informix][Informix ODBC Driver][Informix]The specified table
(nonexistingtable) is not in the database. (SQLPrepare[-206] at
/home/holgerk/php/php-5.1.2.pdo/ext/pdo_informix/informix_driver.c:118)
)

```

---

In addition to using the generic exceptions provided by the PHP 5, you are open to extend the exception class by your own exceptions. For example, you can define different severities for SQL errors. Critical database errors are, for instance, tables which do not exist or the database connections cannot be established. The application will not be able to continue with these critical errors. Non-critical errors, such as locking errors, could be handled by a retry.

Example 5-89 extends the standard exception class with two new database exception classes, and depending on the severity, different actions will be taken.

*Example 5-89 Define your own exceptions with PHP 5 and Informix PDO*

---

```

<?php

/*
own Exception classes for minor and major errors
*/

class CriticalDatabaseErrors extends Exception
{
    public function __construct($message, $code = 0) {
        parent::__construct($message, $code);
    }
}

class NonCriticalDatabaseErrors extends Exception
{
    public function __construct($message, $code = 0) {
        parent::__construct($message, $code);
    }
}

```

```

    }

    try
    {

        $dbh = new PDO("informix:: database=vps_ansi; server=on1000UC3soc;",
            "informix", "123456");

        $stmt=$dbh->exec("SET ISOLATION REPEATABLE READ");
        $stmt=$dbh->query('SELECT * FROM carr');
        $error=$dbh->errorInfo();
        if (!$error[1]) {
            do
            {
                $row=$stmt->fetch(PDO::FETCH_ASSOC);
                $error=$dbh->errorInfo();
                if ($error[1]) throw new NonCriticalDatabaseErrors($error[1]);
            }
            while($row) ;
        }
        else {
            throw new CriticalDatabaseErrors($error[2]);
        }
    }
    catch ( CriticalDatabaseErrors $cde )
    {
        printf("<pre>CritError!: %s \n</pre>", $cde->getMessage()) ;
    }
    catch ( NonCriticalDatabaseErrors $ncde )
    {
        printf("<pre>NonCritError!: %s \n</pre>", $ncde->getMessage()) ;
    }
    catch (PDOExeception $ncde) {
        printf("<pre>Error!: %s \n</pre>", $e->getMessage()) ;
        exit;
    }
}
?>

```

### Output:

```

php_error_exclass.php
<pre>CritError!: [Informix][Informix ODBC Driver][Informix]The specified table
(informix.carr) is not in the database. (SQLPrepare[-206] at
/home/holgerk/php/php-5.1.2.pdo/ext/pdo_informix/informix_driver.c:118)
</pre>

```

---

## Error handling with ifx\_\*

The ifx\_\* interface provides application development the ability to get the database status for the last executed statement by calling the ifx\_error or ifx\_errormsg function. In Example 5-90, we show two error situations: in the first script, the connection fails and in the second script, there is a locking error. In this example, we demonstrate how to use @ operator to suppress error messages.

*Example 5-90 Error codes in ifx\_ with ifx\_error() and ifx\_errormsg()*

---

### Example 1: Connectivity

```
<?php

/*
see the @ at the begin of the line
*/

@$link = ifx_connect("vps99@on1000UC3soc","informix","123456");
if ( !$link ) {
    printf("<pre>");
    printf("Connect: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
    printf("</pre>");
}

@ifx_close($link);

?>
```

Output :

```
<pre>Connect: Returncode E [SQLSTATE=IX 000  SQLCODE=-329] -- Database not
found or no system permission.
</pre>
```

### Example 2: Locking Error

```
<?php

@$link = ifx_connect("vps_ansi@on1000UC3soc","informix","123456");
@$tab_list = ifx_query ("SELECT * FROM car",$link);
if ( $tab_list == 0 ) {
    printf("<pre>");
    printf("Connect: Returncode %d -- %s \n",ifx_error(), ifx_errormsg());
    printf("</pre>");
    exit();
}
```

```

}

do {
    @$row=ifx_fetch_row($tab_list);
}
while( $row ) ;
printf("Connect: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
@ifx_close($link);

?>

Output:
Connect: Returncode E [SQLSTATE=IX 000  SQLCODE=-243] -- Could not position
within a table (informix.car).

```

---

## Error handling with PEAR DB

In the PEAR DB, all calls of the underlying ifx\_\* functions within the objects are quiet. The errors should be handled at the DB error layer. Example 5-91 shows the usage of the status check and the appropriate error routines. The example uses getMessage() and getUserInfo() in parallel to show you the different levels of significance of an error message for the same error.

*Example 5-91 getMessage and getUserInfo in DB*

---

### Example 1: Connectivity

```

<?php

ini_set("include_path","/usr/local/lib/php");
require_once("DB.php");

$dbh = DB::connect("ifx://:@on940soc/stores_demo");

if (DB::isError($dbh))
{
    printf(" Datenbank connection problem \n");
    printf(" getMessage: %s \n",$dbh->getMessage());
    printf(" getUserInfo: %s \n",$dbh->getUserInfo());
}

?>

```

### Output :

```

# php error_conn.php
Datenbank connection problem
getMessage: DB Error: connect failed
getUserInfo: [nativecode=E [SQLSTATE=IX 000  SQLCODE=-25555] Server on940soc
is not listed as a dbserver name in sqlhosts.] ** ifx://:@on940soc/stores_demo

```

### Example 2: Table doesn't exist

```
<?php

ini_set("include_path","/usr/local/lib/php");
require_once("DB.php");

$dbh = DB::connect("ifx://informix:123456@tcp+on1000UC3soc:1526/vps");
$dbh->setFetchMode(DB_FETCHMODE_ASSOC);

$sql="SELECT customer_num FROM carr";
$erg=$dbh->query($sql);
if (DB::isError($erg)) {
    printf(" SQL Error occured gM: %s \n" , $erg->getMessage());
    printf(" SQL Error occured gUI: %s \n" , $erg->getUserInfo());
}
else
$row=$erg->fetchRow();

?>
```

#### Output :

```
# php error_sql.php
SQL Error occured gM: DB Error: no such table
SQL Error occured gUI: select customer_num from carr [nativecode=E
[SQLSTATE=42 000  SQLCODE=-206] The specified table (carr) is not in the
database.]
```

---

#### Note

There is a problem within the PEAR DB package Version 1.7.6 for locking issues. If the query was executed and the first fetch of the query failed with a locking error, this error will not be reported by the package. This will look like the query was executed but no row was found. Fix this problem by adding your own error handling in the ifx.php file. For an example where the error code will be determined and printed, see Example 5-92. When you modify the code, make sure that the syntax of the change is correct. An easy way to check the syntax after your changes is using `php <file>`.

#### Example 5-92 Changes for determining locking errors for SELECTS in PEAR DB

---

Change ifx.php

```
function fetchInto($result, &$arr, $fetchmode, $rownum = null)
{
    .....
}
```

```

if (!$arr = @ifx_fetch_row($result, $rownum)) {
    printf("%s\n",ifx_error());
    return null;
}

```

<-- add here your change

---

## Error handling with unixODBC

In unixODBC, the application has the `odbc_error` and `odbc_errormsg` functions for requesting the status of the last statement. In Example 5-93, we used a connection problem and a permission problem to demonstrate the usage of the mentioned function. The output of the script is also shown in the example.

*Example 5-93 odbc\_error() and odbc\_errormsg() in unixODBC*

---

### Example 1: Connection problem

```

<?php

@$cn =
odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=vps99","informix","1
23456");
if ( !$cn ) {
    printf("<pre>Connect: attempt to connect was not successful \n</pre>");
    printf("<pre>Error   : %d %s\n</pre>",odbc_error(),odbc_errormsg());
    /* exit();*/
}

@odbc_close($cn);
?>

```

#### Output:

```

<pre>Connect: attempt to connect was not successful
</pre><pre>Error   : 0 [unixODBC][Informix][Informix ODBC
Driver][Informix]Database not found or no system permission.
</pre>

```

### Example 2: Permission problem

```

<?php

$cn =
odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=vps_perm","holgerk",
"123456"
);
if ( !$cn ) {
    printf("<pre>Connect: attempt to connect was not successful \n</pre>");
}

```

```

        exit();
    }

    $str="SELECT * from carlot";

    @$odbc_result = odbc_exec($cn,$str);
    if ( !$odbc_result ) {
        printf(" %s %s \n",odbc_error($cn), odbc_errormsg($cn));
        exit();
    }
    @odbc_close($cn);
?>

Output :
37000 [unixODBC][Informix][Informix ODBC Driver][Informix]No SELECT permission.

```

---

## 5.2.8 Transactions and isolation level

Transactions and isolation level provide data consistency in the database server for concurrent working user sessions. Transactions in the client application define which SQL statements belong logically together, and after their execution, if they should be applied or not in the database. Isolation level settings describe the data read behavior in a session in reference to parallel modified data rows and possible repeats of reads. They provide a consistent data view according to the setting in the user session.

### Transactions

Enabling and implementing transactions in the application environment is a divided step from the view of the Informix IDS. On one side, the database administrator has to decide in which logging mode the database should initially be created. There are influences from the performance, portability, parallelism, and data consistency to consider.

On the other side, application development has to look after the consistency of data in the implementation process. Additionally, you need to consider the relationship between transaction management and cursors. Finally, application development has to decide which transaction interface to use. As we will show in the examples, most of the interfaces have their own functions for transaction management. This is related to their understanding that in a default behavior, each statement runs in a single transaction. The term for this is autocommit. So they additionally provide functions to group multiple statements to one transaction. Certainly, the opportunity also exists to use the SQL interface for transactions. But you should not mix both interfaces.

Before we start giving an overview about the available function set for transaction management, we need a short discussion about the available logging modes for databases from the view of the Informix IDS database server. There are four logging modes available. Initially, if nothing else is specified, the database will be created with no logging, but this also means no transactions are available. Different than that, a database can have logging with buffered and unbuffered mode. Logging will take place from the database point of view and in unbuffered mode, each commit will flush the logbuffer of the database server to disk. In buffered mode, the logbuffer will be flushed if it is full. Decision should be made based on available log space and performance impact. The highest log level is log mode ansi, flushing the log buffer at commit time. An additional requirement here is that transactions have to be finished either with rollback or commit.

Unbuffered and ANSI logging are supporting the feature autocommit mode available in most interfaces, which simply means that not each statement will be treated as one transaction.

### ***Transactions in PDO***

We start the overview of available transaction management with a small script using the default SQL interface as shown in Example 5-94. In this example, we take the standard SQL interface using `BEGIN WORK` and `COMMIT` or `ROLLBACK`.

*Example 5-94 Transactions with native SQL in PDO*

---

```
<?php

$dbh = new PDO("informix:: database=vps_log; server=on1000UC3soc;",
"informix", "123456");

$dbh->exec("BEGIN WORK;");

$statement="INSERT INTO car VALUES
(0,'SUV','Temia','1971','Black','Towing','N','Y','N','6',97077,'Sold
as-is.');";

$dbh->exec($statement);
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" execute insert failed with %s \n",$error["1"]);
    $dbh->exec("ROLLBACK WORK;");
    exit(1);
}

$dbh->exec("COMMIT WORK;");
?>
```

---



Additionally, there is an Informix PDO function set for supporting similar functionality as described in Example 5-95. You need to use `beginTransaction` to disable the autocommit mode which is the default behavior in Informix PDO. Calling `commit` or `rollback` will finish the transaction. Do not mix the native SQL function set with the Informix PDO set. It will cause unexpected results.

---

*Example 5-95 Transactions with PDO functions*

---

```
<?php

$dbh = new PDO("informix:: database=vps_log; server=on1000UC3soc;",
"informix", "123456");

$dbh->beginTransaction();

$statement="INSERT INTO car VALUES
(0,'SUV','Temia','1971','Black','Towing','N','Y','N','6',97077,'Sold
as-is.');";

$dbh->exec($statement);
$error=$dbh->errorInfo();
if ( $error["1"]) {
    printf(" execute insert failed with %s \n",$error["1"]);
    $dbh->rollback();
    exit(1);
}

$dbh->commit();
?>
```

---

***Transactions with ifx\_\****

`ifx_*` only supports SQL transaction management. There are no additional functions from the interface. Example 5-96 is a simple test case which uses `ifx_query` to execute the statements.

---

*Example 5-96 Native transactions in ifx\_\* (only available interface)*

---

```
<?php

$link = ifx_connect("vps_ansi@on1000UC3soc","informix","123456");
if (! @ifx_query("BEGIN WORK;",$link) )
{
    printf("Connect: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
}

$statement="INSERT INTO VPS.customer VALUES
(0,'Mrs.','Angela','Clanny','Queen','Ottawa','Saskatchewan','CODE','8337808','4
545622',2329128,'97650',NULL);";
```

```

ifx_query($statement,$link);
printf("Connect: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());

/*
if (! @ifx_query("COMMIT WORK;", $link) )
*/
if (! @ifx_query("ROLLBACK WORK;", $link) )
{
printf("Connect: Returncode %s -- %s \n",ifx_error(), ifx_errormsg());
}
@ifx_close($link);
?>

```

---

### ***Transactions in PEAR DB***

Similar to Informix PDO, PEAR DB supports both ways of maintaining transactions. The native SQL interface is supported by the generic PEAR DB query call. The use of SQL interface for transaction management is shown in Example 5-97.

#### ***Example 5-97 Native transactions in PEAR DB***

---

```

<?php

ini_set("include_path","/usr/local/lib/php");
require_once("DB.php");

$dbh = DB::connect("ifx://informix:123456@on1000UC3soc/vps_ansi");

$result= & $dbh->query ('BEGIN WORK');
if (DB::isError($result)) {
    die($result->getMessage());
}

$startdate=date("m/d/y");
$enddate = date("m/d/y",mktime(0, 0, 0, date("m")+1 , date("d"), date("Y")));
$data = array(0, 1050,0.3,$startdate,$enddate );

$stmt = $dbh->prepare('INSERT INTO promotion VALUES (?,?,, ?,?);');
$erg=$dbh->execute($stmt,$data);
if (DB::isError($erg)) {
    $dbh->query ('rollback work');
    die($erg->getMessage());
}

$result= & $dbh->query ('COMMIT WORK');
if (DB::isError($dbh)) {
    die($dbh->getMessage());
}

```

?>

---

If you do not work with the SQL interface and you have logging databases, the PHP will execute the statements in autocommit mode. This means every statement will be handled as a single transaction. PEAR DB provides a set of functions for combining a set of SQL statements into one transaction. Do not try to mix the SQL and the PEAR DB transaction management. Example 5-98 shows how to use the `autocommit`, `commit`, and `rollback` functions.

---

*Example 5-98 Autocommit with PEAR DB*

---

```
<?php

ini_set("include_path","usr/local/lib/php");
require_once("DB.php");

$dbh = DB::connect("ifx://informix:123456@on1000UC3soc/vps_ansi");

$dbh->autoCommit(false);

$startdate=date("m/d/y");
$enddate = date("m/d/y",mktime(0, 0, 0, date("m")+1 , date("d"), date("Y")));
$data = array(0, 1050,0.3,$startdate,$enddate );

$stmt = $dbh->prepare('INSERT INTO promotion VALUES (?,?,, ?,?);');
$erg=$dbh->execute($stmt,$data);
if (DB::isError($erg)) {
    die($erg->getMessage());
}

$erg=$dbh->rollback();
if (DB::isError($erg)) {
    die($erg->getMessage());
}
$erg=$dbh->execute($stmt,$data);
$erg=$dbh->commit();
?>
```

---

### ***Transactions in unixODBC***

unixODBC also supports both ways of transaction management. Example 5-99 shows the use of the SQL interface with the `odbc_exec` call.

---

*Example 5-99 Native SQL transactions with unixODBC*

---

```
<?php
```

```

$cn =
odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=vps_ansi","informix"
,"123456"
);

if (!@odbc_exec($cn,"BEGIN WORK"))
{
    printf("BEGIN %s %s \n",odbc_error($cn), odbc_errormsg($cn));
}

$parameterlist = array ( 10000,9600, 800, 1000, 11400 ,"cheque");
$str="INSERT INTO neworder VALUES ( ?,?, ?, ?, ? , ?)";
$stmt = odbc_prepare ($cn, $str);
if ( !$stmt ) {
    printf("PREPARE %s %s \n",odbc_error($cn), odbc_errormsg($cn));
    @odbc_exec($cn,"ROLLBACK WORK");
    exit();
}
$odbc_result = odbc_execute($stmt,$parameterlist);
if (!@odbc_exec($cn,"COMMIT WORK"))
{
    printf("COMMIT %s %s \n",odbc_error($cn), odbc_errormsg($cn));
}
?>

```

---

By default, unixODBC runs the SQL statements in autocommit mode. The unixODBC interface provides its own function set for defining the SQL statements which should be run as one transaction. See Example 5-100.

---

*Example 5-100 Autocommit with unixODBC*

---

```

<?php

$cn =
odbc_connect("Driver=Informix;Server=on1000UC3soc;Database=vps_log","informix",
"123456"
);

if (!@odbc_autocommit($cn,FALSE))
{
    printf("BEGIN %s %s \n",odbc_error($cn), odbc_errormsg($cn));
}

$parameterlist = array ( 10000,9600, 800, 1000, 11400 ,"cheque");
$str="INSERT INTO neworder VALUES ( ?,?, ?, ?, ? , ?)";
$stmt = odbc_prepare ($cn, $str);

$odbc_result = odbc_execute($stmt,$parameterlist);
if (!@odbc_rollback($cn))

```

```

{
    printf("ROLLBACK %s %s \n",odbc_error($cn), odbc_errormsg($cn));
}
$odbc_result = odbc_execute($stmt,$parameterlist);

if (!@odbc_commit($cn))
{
    printf("COMMIT %s %s \n",odbc_error($cn), odbc_errormsg($cn));
}
?>

```

---

## Isolation levels in Informix IDS

This section provides a brief overview of the isolation levels and their relationships to transactions and locks. An isolation level setting will influence the behavior of the SELECT statements in a session. The Informix IDS provides four isolation levels:

### ► Dirty read

Dirty read is the default setting for non-logging databases. If this level is set in a user session, the row will be returned to the application since it was read in the server with no check of pending uncommitted changes.

### ► Committed read

Committed read is the default setting for databases created with buffered and unbuffered logging mode. Rows as a result of a SELECT statement will only be returned to the application if those rows are not locked exclusively by a concurrent user session. Reasons for exclusive locks could be pending updates, inserts, or deletes.

### ► Cursor stability

The database server guarantees the committed read behavior for the lifetime of a cursor. The server will not only check pending changes, it will also place a shared lock for every row read by this cursor, preventing concurrent sessions from changing this particular data. The lock will be freed at cursor close or by fetching the next row.

### ► Repeatable read

This isolation level is the default level for databases created with log mode ANSI. It guarantees that every piece of data read during a transaction will remain as read. The database management system has to provide the same data for every attempt to read the same row again in the same transaction. This will be accomplished by putting a shared lock on the data rows read by the session in the database server. The locks will be freed at transaction end.

The correct choice of the isolation level in combination with the lock mode of the base table and the lock wait time are essential for the application in view of parallel use (or the amount of parallel users). For an application which uses a read only database and has only five parallel users, the locks and isolation level are not an important issue. However, for a US Web application which can have zero parallel users at night upto millions in the half time break of a bowl game, the application has to define the correct locking granularity carefully.

Example 5-101 is a simple script to show you how to use isolation levels and how to check settings in the Informix IDS.

---

*Example 5-101 isolation levels with PDO*

---

```
<?php
$dbh = new PDO("informix:: database=vps_log; server=on1000UC3soc;",
"informix", "123456",array(PDO::ATTR_PERSISTENT=> true));

$dbh->exec("set isolation to repeatable read");
?>
```

run **onstat -g sql** for verification

```
IBM Informix Dynamic Server Version 10.00.UC3R1  -- On-Line -- Up 1 days
03:28:44 -- 86636 Kbytes
```

Sess Id	SQL Stmt type	Current Database	Iso Lock Lv1 Mode	SQL ERR	ISAM ERR	F.E. Vers	Explain
185	-	vps_log	RR Not Wait	0	0	9.03	Off

Look at the IsoLv1 column, RR means here Repeatable Read.

---

Following are a few performance considerations for databases with log mode ANSI and default isolation level repeatable read:

- ▶ Try to avoid sequential scans with repeatable read which will lock the whole table with a shared lock, preventing others from any changes.
- ▶ Try to set the table lock mode to row to lock only the rows which are actually read. Depending on the row size of the table, lock mode page may lock many more rows than were actually read.
- ▶ Run update statistics periodically to speed up index reads wherever applicable.
- ▶ Keep transactions small.
- ▶ Avoid setting lock mode to not wait.

## 5.2.9 PHP and Informix XPS SQL extensions

Informix XPS is the database server product developed for maintaining and processing huge amounts of data in a data warehouse environment. This product is based on a shared nothing architecture. Think of it as a logical group of multiple database instances working together. This group can be physical on one machine, or on several machines tied together with an high speed connection. The primary benefit for this architecture is the load balancing of disk I/O and CPU, which is achieved by an improved fragmentation functionality and special design for data exchange between the instances. There are several additional features in the data administration area in comparison with Informix IDS. This section cover some features provided by the SQL layer which could be very interesting from an application development point of view.

The features described in PHP examples are:

- ▶ External tables for loading large amount of data
- ▶ Sysdbopen and sysdbclose store procedures for housekeeping
- ▶ MERGE statement for an easy implementation of “upserts”

### External tables

External tables are an easy to use SQL extension for loading and unloading large amounts of data into and from the database. The data source can come from one or multiple pipes or flat files on the disk produced by an host application or different database instance of various vendors.

Example 5-102 shows how to create an external table and load data into the CAR table using the PDO interface.

*Example 5-102 External table with PDO*

---

```
<?php

try {
$dbh = new PDO("informix:host=Linux; service=11005; database=vps;
server=on850soc; protocol=onsoctcp;", "informix", "123456");

/*
drop and recreate the table car
*/

$dbh->query ( "drop table vps.car; ");

$dbh->query ( "CREATE RAW TABLE vps.car ( t_id serial (1000) , t_type
varchar(50) not null, t_model varchar(50) not null, t_year integer, t_color
varchar(50), t_package varchar(50), t_ac char(1) check (t_ac in ('Y','N')),
t_powerwindows char(1) check(t_powerwindows in ('Y','N')), t_automatic char(1)
```

```

check(t_automatic in ('Y','N')), t_doors smallint check(t_doors > 1), t_price
decimal(10,2) not null check(t_price >=0), t_data varchar(200));");

/*
create the external table here , use one flat file for load
*/

$dbh->query ( "CREATE EXTERNAL TABLE car_external SAMEAS car USING (
DATAFILES(\"disk:1:/home/vps/informix/CAR.TBL.001\")");

/*
load the data here
*/
$dbh->query ("INSERT INTO car SELECT * FROM car_external");

$error=$dbh->errorInfo();
print_r($error);

} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    exit;
}
?>

```

---

## Sysdbopen and sysdbclose stored procedures

A very common requirement for a database server is to allow the administrator to add hidden actions at the beginning and the end of an user connection in the database server. These actions could be related to default settings for the user, such as roles, isolation levels, memory settings, optimizer hints, or tracking activities. Similar activities are sometimes required at the end of an user session, such as deleting session specific tables or tracking accounting information. Informix XPS provides two stored procedures to serve these purposes:

- ▶ Sysdbopen automatically executed at connection time for a user
- ▶ Sysdbclose automatically executed at connection close time for a user

A very common case using an *sysdbopen* stored procedure, which also is really useful for our car dealer application, is related to security. We have two sets of permissions in the database because a user could be acting as a car dealer or a (potential) customer. From a security design standpoint, it makes sense to create a customer role and a dealer role. Both roles have to be granted the necessary permission set for the base tables or views. Each new user created in the database has to be granted only a role, not the whole permission set for all base tables.



Example 5-103 provides you with a general concept of how to use sysdbopen and sysdbclose for tracking user activities. Besides defining some global user session settings, the sysdbopen procedure will also insert a new row in a tracking table. Sysdbclose will use a global variable in the procedure and update the previously inserted row to set the logout time. Here we use the procedures for a specific user. Sysdbopen and sysdbclose can be created for public or for various users in parallel.

*Example 5-103 sysdbopen and sysdbclose for accounting and initial settings*

---

```
/* database has the following table

create table tracking (
id serial(100),
username char(30),
logintime datetime year to second,
logouttime datetime year to second
)

/* create the following sysdbopen sp for user1 */

CREATE PROCEDURE "user1".sysdbopen()
DEFINE sessionid int ;
DEFINE usern      char(30);
DEFINE global tracking int default 1;

-- Tracking

let usern="";
let sessionid =0;

SELECT dbinfo("sessionid") INTO sessionid FROM systables WHERE tabid=1;
SELECT username INTO usern      FROM sysmaster@on850soc:sysessions
WHERE sid=sessionid;

INSERT INTO tracking VALUES ( 0,usern      , current, NULL ) ;
-- get the serial value from the last insert
SELECT dbinfo("sqlca.sqlerrd1") INTO tracking FROM systables WHERE tabid=1;

-- initial settings for directives, PDQ , isolation level
SET ENVIRONMENT hash_join ON;
SET PDQPRIORITY 2;
SET ISOLATION TO COMMITTED READ;

END PROCEDURE;

/* create the following sysdbclose sp for user1 */
```

```

CREATE PROCEDURE "user1".sysdbclose()
DEFINE global tracking int default 1;
UPDATE tracking SET logouttime=current
WHERE id=tracking;
END PROCEDURE;

```

Output in the tracking table after disconnect :

```

id          100
username    user1
logintime   2006-02-07 10:13:36
logouttime  2006-02-07 10:13:41

```

---

Example 5-104 shows a way to simplify the usage of roles in a sysdbopen. The prerequisite here is that each user will connect to the database server as the specific user.

*Example 5-104 sysdbopen with setting roles*

---

```

CREATE PROCEDURE "holgerk".sysdbopen()
DEFINE sessionid int ;
DEFINE count int ;
DEFINE usern      char(30);
DEFINE global tracking int default 1;

--SET DEBUG FILE to "/tmp/sysdbopen_role" ;
--TRACE ON;

LET usern="";
LET sessionid =0;
LET count =0;

SELECT dbinfo("sessionid") INTO sessionid FROM systables WHERE tabid=1;
SELECT username INTO usern      FROM sysmaster:syssessions
WHERE sid=sessionid;

SELECT count(*) INTO count FROM dealer WHERE c_name=usern;
IF (count>=1) THEN
SET ROLE dealer;
ELSE
SET ROLE customer;
END IF

END PROCEDURE;

```

---

## MERGE statement

In the Dealership application, the dealer will need to update the car inventory table from time to time. When updating the CAR table, the action will be INSERT for a new car and an UPDATE for an existing car. The MERGE statement can be used for this purpose.

Since the MERGE SQL statement will not update data in the third table, it cannot be used if we keep the inventory in a separate table as the original car dealer data model. To merge the new cars into the existing stock, we have to change the schema of the CAR table. Example 5-105 is the PHP script for Informix PDO using the MERGE statement. In this example, we added to the CAR table the `n_quantity` column which shows the number of cars in stock. The script first creates a new car table `CAR_NEW` with the additional quantity column. The new shipment is loaded into the new car table. The new car table and the old car table then are merged. During the merge, if a car exists, then `n_quantity` will be increased by 1, if not, the new car will be inserted.

---

### *Example 5-105 Merge and Informix PDO*

---

```
<?php

try {
$dbh = new PDO("informix:host=Linux; service=11005; database=vps;
server=on850soc; protocol=onsoctcp; ", "informix", "123456");

    $dbh->query ( "DROP TABLE car_new; ");

    $dbh->query ( "CREATE TABLE CAR_new ( t_id serial (1000) , t_type
varchar(50) not null, t_model varchar(50) not null, t_year integer, t_color
varchar(50), t_package varchar(50), t_ac char(1) check (t_ac in ('Y','N')),
t_powerwindows char(1) check(t_powerwindows in ('Y','N')), t_automatic char(1)
check(t_automatic in ('Y','N')), t_doors smallint check(t_doors > 1), t_price
decimal(10,2) not null check(t_price >=0), t_data varchar(200), n_quantity
integer);");

    /*
create the external table here
*/

    $dbh->query ( "CREATE EXTERNAL TABLE car_external SAMEAS car USING (
DATAFILES(\"disk:1:/home/informix/newcars.unl\"))");

    /*
load the data here
*/
```

```

$dbbh->query ("INSERT INTO car_new SELECT * FROM car_external");

/*
merge the data into the final table
*/
$dbbh->query ("MERGE INTO car old USING car_new AS new on old.t_id=new.t_id
WHEN MATCHED THEN UPDATE SET old.n_quantity=old.n_quantity+1 WHEN NOT MATCHED
THEN INSERT VALUES (new.t_id,new.t_type,new.t_model,
new.t_year,new.t_color,new.t_package
,new.t_ac,new.t_powerwindows,new.t_automatic,new.t_doors,new.t_price,
new.t_data,new.n_quantity);");

$dbbh->query ( "DROP TABLE car_new; ");

} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    exit;
}
?>

```

Given there are actually 2 cars in the car table,  
select t\_id, n\_quantity from car

t_id	n_quantity
1011	1
1012	1

After than we merge 3 new and additional 2 existing cars together, the result  
will be :

select t\_id, n\_quantity from car

t_id	n_quantity
1011	2
1012	2
1026	1
1015	1
1014	1

---

Also, the pseudo update in the MERGE statement will not trigger an update trigger to change the inventory table. Example 5-106 provides another solution for the car inventory update from a file. Different than what was done in Example 5-105, the data model does not change. The PHP program reads the data from the file and calls a stored procedure in the database server. The procedure first checks if the row already exists in the table. If not, it will insert a new row into the car table and inventory table. Otherise, it only updates the quantity of the car in the

inventory table. When you use the stored procedure, no table schema change is required.

*Example 5-106 Merge solution/PEAR DB and a stored procedure in the Informix IDS*

---

**PHP Script**

```
<?php

$DEBUG=1;
$DEALER=1;

ini_set("include_path","/usr/local/lib/php");
require_once("DB.php");

$dbh = DB::connect("ifx://informix:123456@on1000UC3soc/vps2");

$handle = fopen ("CAR.TBL.001", "r");
if ($handle) $buffer = fgets($handle, 4096);
if ($buffer ) $arr=explode("|", $buffer);

$spstatement="execute procedure insert_car ( ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, $DEALER, $DEBUG)";
$stmt = $dbh->prepare($spstatement);
if (DB::isError($stmt)) {
    die($stmt->getUserInfo());
}
do {
    $arr=explode("|", $buffer);
    unset($arr[12]);
    $dbh->execute($stmt,$arr);
    if (DB::isError($stmt)) {
        die($stmt->getUserInfo());
    }
    $buffer = fgets($handle, 4096);
}
while (!feof($handle) && $buffer);
fclose ($handle);
?>
```

**SQL statements for the Stored Procedures**

```
dbaccess << EOF
database vps;

drop procedure insert_car;
create procedure insert_car (
m_id integer,
```

```

m_type varchar(50),
m_model varchar(50),
m_year  varchar(4),
m_color varchar(50),
m_package varchar(50),
m_ac varchar(1),
m_power varchar(1),
m_automatic varchar(1),
m_doors integer,
m_price decimal(12,2),
m_data varchar(200),
dealer integer,
DEBUG integer
)

define local_m_id integer;
define local_m_type char(50);
define local_m_model char(50);
define local_m_year integer;
define local_m_color char(50);
define local_m_package char(50);
define local_m_ac char(1);
define local_m_power char(1);
define local_m_automatic char(1);
define local_m_doors integer;
define local_m_price decimal(12,2);
define count integer;

if DEBUG = 1 then
set debug file to "/tmp/insert_cars";
trace on;
set explain on;
end if

let local_m_type = trim(m_type) ;
let local_m_model = trim(m_model);
let local_m_year = m_year ;
let local_m_color = trim(m_color) ;
let local_m_package = trim(m_package) ;
let local_m_ac = trim(m_ac) ;
let local_m_power = trim(m_power) ;
let local_m_automatic = trim(m_automatic);

select t_id,count(*) into local_m_id,count
from car
where
t_type = local_m_type and

```

```

t_model = local_m_model and
t_year::int = local_m_year and
t_color = local_m_color and
t_package = local_m_package and
t_ac = local_m_ac and
t_powerwindows = local_m_power and
t_automatic = local_m_automatic and
t_doors = m_doors and
t_price = m_price
group by 1;

if count > 0
then
update inventory set n_quantity = n_quantity+1
where f_d_id=dealer and f_m_id=local_m_id;
else
insert into car values (
0, m_type , m_model , m_year , m_color , m_package , m_ac ,
m_power , m_automatic , m_doors , m_price ,m_data );
let local_m_id=dbinfo("sqlca.sqlerrd1");
insert into inventory values ( dealer,local_m_id,1 );
end if

end procedure;
EOF

```

---

## 5.3 Optimization

Optimizing the SQL statements should be part of the development cycle of the database-based application. This can be part of the QA process at the end of the development process, but could also be included after you finish certain parts of the new application. The major focus of the optimization is minimizing the execution time. Logical correctness and memory consumption should be checked. The number of parallel user sessions should also be considered.

In this section, we focus on the discussion about optimization of the SQL statements used by the application and the Informix database server.

### Application tuning

On the application side, SELECT statements and the stored procedures are the most common source of performance issues. If you measure the run time of all the SQL statements and come up a top 10 long running statement list, we expect most of the time you will see SELECT in that list. The list can be created by simply adding time measurements into the scripts or using external software, such as Informix Ispy, which would allow you to figure out the frequency of statements

and their run times. In this section, we focus on how to check if the slow running SQL can be improved.

We have compiled a small checklist with the appropriate action for the most common reasons of slow queries.

### ***Checking single query***

- ▶ Is this query really doing what you expect it to do?
  - Check the logical correctness
  - Review the filters in the where clause
  - Review all tables which will be joined, avoid Cartesian products
  - In the case of a join (common in a n:m relationship), make sure that the intermediate result is not too large to cause hash table overflow.
- ▶ Does the query use any indexes on the base tables?
  - Do indexes exists on the table?  
Run dbschema against the database and the involved tables.
  - Do you really use indexed fields in your query?  
Also here dbschema would be helpful.
  - Query uses indexed fields but answer time is slow.  
Use the SET EXPLAIN ON statement to check the query plan. Check the generated sqexplain.out file. Use the Optimizer directives to force the best plan that you think is the best choice and compare with the system chosen plan. Run update statistics with the medium option to generate distributions for your column.
  - Are the statistics up to date?  
If you use medium or high in your statistics, check the sysdistrib system table. Run the following select, check if there is a result set and if yes, check the timestamp of the created column. In the case of no output or a very old timestamp, create a new statistic on the table:  

```
SELECT tabname, created FROM systables, sysdistrib  
WHERE systables.tabid=sysdistrib.tabid and tabname="<tablename>"
```

  
Another simple way of checking is run the query, then run onstat -g dsc and look for columns from your table.  
If you normally run update statistics without medium/high mode, check if there is a statistic available for your table with the following SQL:  

```
SELECT nrows, npused FROM systables where tabname="<tablename>"
```



The values for columns in the result should not be 0 and they should match with an oncheck -pt output for the same table.

- ▶ Does the query use multiple tables in the join?

- Does the optimizer use the right join order?

Check this again with the sqexplain.out created by the SET EXPLAIN statement. If you find out that the order has room for improvement, run a update statistics medium/high on the join columns.

### ***Checking stored procedures***

- ▶ Check the creation date of the plan of the stored procedure.

Use the following SQL to verify the creation date:

```
SELECT a.procid,a.procname, b.created FROM sysprocedures a,  
sysprocplan b where a.procid=b.procid and procname="<procedurename>";
```

- ▶ Check for unnecessary iterations and the depth of the call hierarchy.

Check this with onstat -g ses <your sessionid>

- ▶ Check the stored procedure variable settings for logical correctness.

Use TRACE ON and SET DEBUG FILE. For details, see 5.2.5, “Working with stored procedures and user defined functions” on page 276.

- ▶ Do you see many locking errors in attempt to execute procedures?

- Check the stored procedure logic, and try to avoid temp tables in the procedure.
  - Set up a schedule for running statistics update.
  - Avoid changing the DDL of the base table for the stored procedure frequently.

### ***Checking multiple users in parallel***

- ▶ Do you see locking errors such as -243, -244, or -245 with ISAM error 113 or 107?

- Check application lock mode settings; increase the wait time.
  - Check table lock mode settings on the database server; alter the table lock mode to row.
  - Check the length of transactions in your applications for the major base tables.
  - In the case of ANSI databases:
    - Check the statistics for the tables with locking errors.
    - Add indexes for the filter columns you mainly use.

- Do you see errors such as -208 or -406, or do your applications hang for long wait times for the database server?
  - Check the settings of your PDQ environment; check especially MAXPDQPRIORITY, DS\_TOTAL\_MEM, and DS\_MAX\_QUERIES in the onconfig file and PDQPRIORITY in the client shell environment.
  - In the case of -208 and -406 errors, increase DS\_TOTAL\_MEM within the scope of your hardware.

## Server tuning

The Informix IDS server environment and the database server configuration are other areas for optimizing. We will not discuss here investigating operating system issues, such as I/O bottlenecks, memory problems, and processor utilization. We focus on the Informix IDS configuration. An complete discussion of all aspects of the Informix IDS optimization is beyond the scope of this book.

## Server utilization

Before we start any tuning activity on the server, we check the utilization of the server first. Keep in mind that performance tuning is not a check of one snapshot of the server status. Any changes in the server environment should be based on at least multiple reviews of the parameter behaviors, their growth, and values. Starting point for the server utilization check is the `onstat -g ath` taken multiple times. Check the output in addition with several `onstat -g act`. If you see all the time only `sqlexec` threads in a status condition `wait (netnorm)` (for sessions coming in over the network) or condition `wait (sm_read)` (in case the Web server and the database server are on the same machine using shared memory for communication), you should again start checking the logic of the application clients, because the server is definitely waiting for work. A sample snapshot for an empty database server is shown in Example 5-107:

*Example 5-107 onstat -g ath with server threads waiting for client activity*

---

```
#onstat -g ath | grep sqlexec
Threads:
  tid      tcb      rstcb    prty status      vp-class    name
.....
  1757     489dc5f0 47e152d8 2    cond wait    netnorm     1cpu       sqlexec
  1759     489f48d0 47e133d0 2    cond wait    netnorm     1cpu       sqlexec
  1761     489f4a18 47e14354 2    cond wait    netnorm     1cpu       sqlexec
  1763     48a145f0 47e12ea4 2    cond wait    netnorm     1cpu       sqlexec
  1765     489f4ca0 47e13e28 2    cond wait    netnorm     1cpu       sqlexec
  1767     48a14828 47e138fc 2    cond wait    netnorm     1cpu       sqlexec
  1769     48a14b00 47e15804 2    cond wait    netnorm     1cpu       sqlexec
  1771     48a14dd8 47e1244c 2    cond wait    netnorm     1cpu       sqlexec
  1773     489e46e0 47e114c8 2    cond wait    sm_read     1cpu       sqlexec
  1775     489e4a08 47e14dac 2    cond wait    sm_read     1cpu       sqlexec
```

1777	489e4d30	47e14880	2	cond wait	sm_read	1cpu	sqlxec
1779	488a4280	47e12978	2	cond wait	sm_read	1cpu	sqlxec
1781	488a45f0	47e15d30	2	cond wait	sm_read	1cpu	sqlxec
1783	48b00880	47e1625c	2	cond wait	sm_read	1cpu	sqlxec
1785	48b234e0	47e16788	2	cond wait	sm_read	1cpu	sqlxec

### Sequential scans from the server view

After we have verified that the server requests are high, another good starting point is investigating the statistics of the server. This only makes sense if the server has been running for a long time, and your application environment was set up and behaved in a way that dissatisfies you. Example 5-108 shows an output of `onstat -p`.

Example 5-108 `onstat -p` and sequential scans

```
IBM Informix Dynamic Server Version 10.00.UC3R1  -- On-Line -- Up 14 days
05:42:28 -- 86636 Kbytes

Profile
dskreads  pagreads  bufreads  %cached dskwrits  pagwrits  bufwrits  %cached
30004      42940      4362191  99.32   88446     143315    928701    90.48

isamtot    open      start     read      write     rewrite   delete    commit    rollbk
3134580    60953     619195    1146627   415510    2444      7658      3343      1923

gp_read    gp_write   gp_rewrt   gp_del    gp_alloc   gp_free    gp_curs
24          8          131        0          0          0          2

ovlock     ovuserthread ovbuff      usercpu    syscpu     numckpts    flushes
0           0          0          112.44     47.34      331         8214

bufwaits   lokwaits   lockreqs   deadlks    dltouts    ckpwait    compress    seqscans
183        0          4843097    0           0          11         4058        34339

ixda-RA    idx-RA     da-RA      RA-pgsused lchwaits
121        21         459        592         2754
```

### Seqscans

This value is proof that your activities in application optimization were successful. Check if this count is high and review the growth of this value over a certain time, when your application is running. If the number is high and a linear growth is visible here, you should review your application again for candidates using sequential scans. Of course there are always queries running without any index and they are fine, but in an OLTP environment, there should not be too many of them.

## Rollbacks

`onstat -p` also gives you an idea if the application works the way you expect. An easy hint that something is wrong in your environment is the value of rollbacks. If this value regularly shows a linear growth at a high level, check your application for unintended rollbacks. Make sure that your open transactions are really committed at the end of your program.

## Buffer

Buffer pool utilization should be analyzed for performance tuning. Check the buffer cache ratio in the output of `onstat -p`. The number should be at least 80%. A lower number indicates that the buffers in memory are insufficient. Another indication of the buffer pool bottleneck are the values of “Fg writes” from an `onstat -F`. See Example 5-109. If you see that this number is in the high two digits, you should increase the number of buffers in your buffer pool.

### Example 5-109 `onstat -F` and the IO statistics

---

```
#onstat -F
Fg Writes      LRU Writes      Chunk Writes
210            22984            1839518

address flusher state  data
47e10544 0      I      0      = 0X0
47e10a70 1      I      0      = 0X0
47e10f9c 2      I      0      = 0X0
47e114c8 3      I      0      = 0X0
47e119f4 4      I      0      = 0X0
states: Exit Idle Chunk Lru
```

---

## Checkpoint frequency and length

As the last of the easy to check factors in the database server, we will discuss the checkpoint length and frequency. Checkpoints periodically happen to synchronize the logs and the data between memory and the disk. During checkpoint time, all sessions' write activities are frozen, only reads are allowed. If the checkpoint happened too often because of too much activity in the server or physical log files are too small, the system will be blocked more often. You should try to avoid the situations as shown in Example 5-110.

### Example 5-110 Checkpoints that are long or happening too frequently

---

```
Checkpoints happened too often
10:51:10 Checkpoint loguniq 5, logpos 0x231088, timestamp: 0x175640
10:51:15 Checkpoint Completed: duration was 8 seconds.
10:51:15 Checkpoint loguniq 5, logpos 0x5e0018, timestamp: 0x1be9c0
10:51:28 Checkpoint Completed: duration was 6 seconds.
10:51:28 Checkpoint loguniq 5, logpos 0x7f1018, timestamp: 0x1beac0
```

```
10:51:58 Checkpoint Completed: duration was 6 seconds.  
10:51:58 Checkpoint loguniq 6, logpos 0x2a2018, timestamp: 0x1f9090  
10:52:10 Checkpoint Completed: duration was 3 seconds.  
10:52:10 Checkpoint loguniq 7, logpos 0x160018, timestamp: 0x2673f0  
10:52:38 Checkpoint Completed: duration was 10 seconds.  
10:53:38 Checkpoint loguniq 7, logpos 0x2f1088, timestamp: 0x2874f0
```

#### **Long Checkpoints**

```
11:04:07 Level 0 Archive started on work  
Wed Oct 19 11:05:07 2005  
11:05:07 Checkpoint Completed: duration was 86460 seconds.
```

---

The duration of the checkpoint depends on the amount of I/O the server has to write from the buffer pool to the disk for physical logs. If you noticed in the log file or with `onstat` that the checkpoint takes a long time, this is an indication that either the physical log is too small, or the amount of dirty buffers between two subsequent checkpoints is too high. Use `onstat -l` to check the size of the physical log. In the case of too many dirty buffers, check the I/O throughput on your disks. Lowering the maximum for dirty buffers defined by `LRU_MAXDIRTY` and `LRU_MINDIRTY` in the `onconfig` file can also help.

## **5.4 Apache, PHP, and Informix IDS on Windows**

After the detailed discussion about developing PHP application with various Informix IDS database interfaces in Linux, we want to give you a short overview about the environment setup on Windows. This section gives detailed information for download, installation, and configuration of Apache and PHP with Informix IDS.

### **5.4.1 Installation and configuration of Apache Web server**

For the installation of the Apache Web Server Version 1.3 or 2.0.55 under Windows, follow these steps:

1. Download the installation package from:  
<http://httpd.apache.org/download.cgi>
2. Unzip the installation file.
3. Execute the installation file.
4. Follow the installation steps from the installer: Use typical installation; provide details for the server name of your Windows machine, administrator account, and domain name.

5. Verify the successful installation of the Web server with a browser and type the server name of your machine. The default Congratulations Web page of Apache should appear.

If you have not redefined the installation directory, the Web server is installed under `c:\Program Files\Apache Group\Apache2`. The Apache server will run as a service and will start automatically. The installation routine will automatically configure all the necessary initial variable settings in the `httpd.conf` file.

## 5.4.2 Installation of the PHP and configuration with Apache

For the installation and configuration of the PHP 5.1 under Windows with the Apache Web server, you need to do the following steps:

1. Download the PHP binaries from the following Web site:  
<http://www.php.net/downloads.php>  
Choose the zip package, not the installer for the download.
2. The downloaded file is a zip of the destination installation directory. For the installation of the PHP, you have simply to extract the file into your final directory.
3. For the configuration of the PHP with Apache, edit the `httpd.conf` file in the configuration directory under the installation directory of your Apache Web server. Add the lines similar to the lines in Example 5-111 to this file. In this example, the PHP was installed in `c:\php`.

*Example 5-111 Settings for PHP in the `http.conf` file for Apache V2 under Windows*

---

```
# For PHP 5 do something like this:
LoadModule php5_module "c:/php/php5apache2.dll"
AddType application/x-httpd-php .php

# configure the path to php.ini
PHPIniDir "C:/php"
```

---

4. For configuration of the PHP, go in your PHP installation directory. Copy the file `php.ini-recommended` to `php.ini`. Make appropriate changes for your environment. If you want to add further extensions, such as the PHP base binaries, you have to add the `extension_dir` parameter and the needed extensions. The extensions can be downloaded from the same download site. In addition, to check the database connection, the parameter `display_errors` should be set to `On`. This setting allows the display of connection errors in the browser. In a production environment, errors should not be reported to the user in a raw format. Set `display_errors` to `Off` is recommended.
5. Verify your changes with:

- Restart the Apache Web server, use **Settings** → **Control Panel** → **Administrative tasks** → **Services**
  - Click on the Apache service which should show the status running. The options Restart and Stop should appear.
  - Click **Restart**.
  - In case of an error, check your newly added settings in the httpd.conf file or check the error file in the logs directory under the Apache installation directory.
- After a successful restart, check the PHP configuration with a small script using the function `phpinfo()`. You can also use the script from Example 5-11 on page 205. If the link of PHP and Apache is successful, you will see a HTML with a large table which describes the global PHP settings and all included PHP extensions and their settings.

### 5.4.3 Informix IDS, Informix connectivity, and PHP

The PHP installed in the previous section statically includes an extension `unixODBC` for accessing an ODBC source, which has to be configured for each database server. For the configuration of the ODBC source and the subsequent connection to the Informix IDS, you need either an Informix SDK or an Informix Connect installed on the Web server machine. If you do not plan to develop applications with ESQL/C, Informix Connect is sufficient. For the installation of the connectivity for the Informix IDS, you have to do the following steps:

1. Download the Informix Connect 2.90.TC3 product from:  
<http://www-306.ibm.com/software/data/informix/downloads.html>
2. Unzip the installation files into a directory.
3. Execute the Setup executable in the unzip directory for installation.
4. The product will be installed in the C:\Program Files\IBM\Informix\Connect directory.
5. In the system environment (**Settings** → **Control panel** → **System**), set the `INFORMIXDIR` environment variable to the installation directory.
6. Restart the Apache service.
7. Verify the Apache environment that `$INFORMIXDIR` is set with `phpinfo()`.
8. If the variable setting does not appear in the `phpinfo` HTML table, reboot the machine.
9. Verify the setting after restarting again.

After installing the Informix Connect, you have to add an ODBC data source to the Windows ODBC manager. You have to do the following steps:

- ▶ Start the ODBC manager from **Settings** → **Control Panel** → **Administrative Tools** → **Data Sources**.
- ▶ Create a new system DSN data source.

To enable the Apache Web server to find the DSN, the DSN cannot only be known by the local user. You have to choose the system DSN because the Apache will run as a service.

Figure 5-7 on page 350 shows you an example window for how to fill out the general section for the data source.

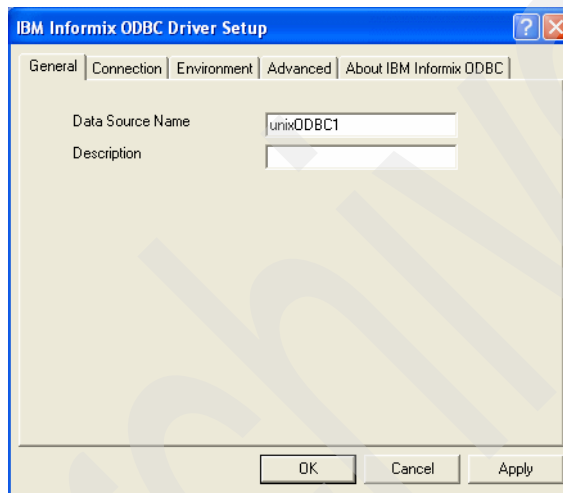


Figure 5-7 General section for ODBC settings

Figure 5-8 shows you the example settings for the connection settings for the data source. You have to change the values for the appropriate settings in your environment.



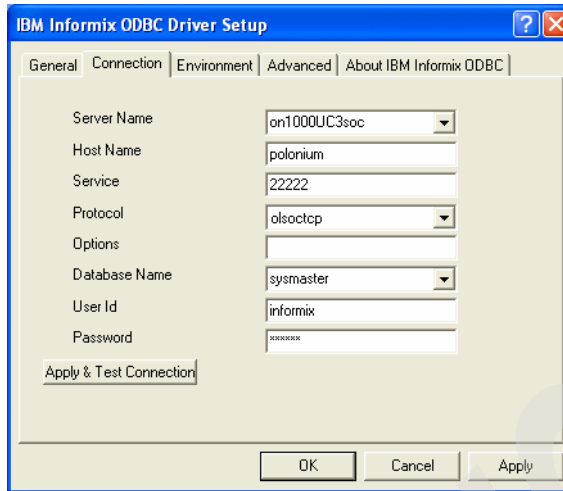


Figure 5-8 IBM Informix ODBC Driver Setup Connection

After setting up the environment and the configuration of the data source, you have to verify the connectivity. You can use the PHP script from Example 5-112 for the verification. This script will use the `sysmaster` database for a select of existing tables. If successful, two rows will be returned.

*Example 5-112 unixODBC connection in a Windows environment*

```
<?php

/* Connection string is different from the string for LINUX !! */
$cn = odbc_connect("DSN=unixODBC1;", "informix", "123456");

if ( !$cn ) {
    printf("<pre>Connect: attempt5 to connect was not successful\n</pre>");
    exit();
}
$str="select * from systables";

$odbc_result = odbc_exec($cn,$str);
if ( !$odbc_result ) {
    printf(" %s %s \n",odbc_error($cn), odbc_errormsg($cn));
    exit();
}

printf("<pre>");
odbc_fetch_row($odbc_result);
```

```
printf(" %s %s %s \n",odbc_result($odbc_result,1), odbc_result($odbc_result,2),
odbc_result($odbc_result,3));

odbc_fetch_row($odbc_result);
printf(" %s %s %s \n",odbc_result($odbc_result,1), odbc_result($odbc_result,2),
odbc_result($odbc_result,3));

printf("</pre>");
@odbc_close($cn);
?>
```

---

Once the initial connection test was successful, you can start your PHP application development process. For a detailed discussion of the use of the unixODBC extension in combination with the Informix IDS database server, see 5.2, “Application development with Informix IDS” on page 227. The scripts for unixODBC we introduced there for the discussion of the major programming concepts can also be used in the Windows environment.

## Port PHP applications from MySQL V5 to DB2 UDB V8.2

MySQL became popular when Internet Service Providers (ISPs) discovered that MySQL could be offered at no charge to their Internet customers, providing all the storage and retrieval functionality a dynamic Web application needs. However, as a customer's business grows, a full function database management system, such as DB2, becomes essential for the success of their business. Converting the database and application to DB2 is a natural choice.

The IBM Redbook, *MySQL to DB2 UDB Conversion Guide*, SG24-7093, contains a detailed discussion about porting a MySQL database, database objects, and applications from MySQL V4 to DB2 UDB.

This chapter focuses on the steps required to convert PHP applications from MySQL V5 to DB2 UDB V8.2. We also cover the conversion of database objects introduced in MySQL V5, including stored procedure, triggers, and user defined functions.

## 6.1 Introduction

Converting from MySQL to DB2 requires proper planning, whether you perform the conversion on your own or use IBM resources. Unexpected delays in project execution can be avoided by following well-planned conversion procedures.

- ▶ Preparation

This includes project planning, education of database administrators and developers, and the setup of the environment.

- ▶ Conversion

This includes the conversion of the database structure, objects, and data, as well as the applications that connect to the database, and any maintenance/batch scripts.

- ▶ Post-conversion

This includes performance tuning of the converted database and application.

The IBM Web site, Porting to DB2 UDB, contains an overview of the porting process and many porting resources:

<http://www-128.ibm.com/developerworks/db2/zones/porting/>

### 6.1.1 IBM migration offering

IBM has significant experience performing database migrations. This know-how is offered to everybody in the form of no-charge promotions as well as low-priced software licenses and services.

#### **DB2 promotion**

*Migrate Now!* for DB2 Universal Database facilitates the migration from MySQL, Microsoft SQL Server, Oracle, Sybase, and additional database platforms to DB2 at a special price with a significant discount. *Migrate Now!* is an end-to-end offering that includes:

- ▶ Migration tool kits
- ▶ Free online education
- ▶ Sales teams and resources to assist you to plan and implement your migration based on IBM's proven methodology

Take advantage of this special offer and Migrate Now! For more information about this promotion, refer to the following Web site:

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

or get in contact with your IBM representative.

## DB2 migration services and support

IBM uses a proven migration methodology which helps reduce costs and risks associated with a migration while addressing its major components:

- ▶ Applications
- ▶ Database design
- ▶ Data movement

Migration consultants can advise your organization regarding a phased migration approach. This approach includes:

- ▶ Assessment of the database conversion effort.
- ▶ DB2 migration assessment.
- ▶ Installation of DB2 and other products you may require.
- ▶ Pilot migration.
- ▶ Full migration of remaining data and applications.

IBM migration specialists have provided advice and counsel to over 70 customers and performed over 3 500 migrations to DB2 from non-IBM database systems, such as Microsoft SQL Server, Oracle, Sybase, MySQL, ProgreSQL, ADABAS, IDMS, SUPRA, TOTAL, CA-IDMS, CA-Datcom and VSAM. If you would like IBM to perform your migration, send an e-mail to the appropriate contact on the following Web site:

<http://www.ibm.com/software/solutions/softwaremigration/contacts.html>

## Technical training

IBM offers a workshop designed for ISVs and IBM Business Partners who intend to sell applications on DB2, and who are new to DB2. It begins with building fundamental DB2 skills, where participants learn how to migrate their existing database to DB2. Participants actually bring their own database to the workshop to migrate. In some cases (depending on database complexity), attendees have been able to completely migrate their database within the workshop duration. Detailed information about the DB2 Enablement Workshops can be found at:

[http://www.developer.ibm.com/spc/events/db2\\_en.html](http://www.developer.ibm.com/spc/events/db2_en.html)

## Contacts

IBM services are available to assist you during any part of the migration. If you have a migration project in mind or you want to obtain the latest conversion information (for example, porting guides), contact:

- ▶ In North America and Latin America, <mailto:db2mig@us.ibm.com>
- ▶ In UK, Europe, Middle East and Africa, <mailto:emeadbct@uk.ibm.com>
- ▶ In Japan, India and Asia Pacific, <mailto:APDB2@nz1.ibm.com>

More information about the DB2 migration team can be found at the *Software Migration Project Office (SMPO)* Web site:

<http://www.ibm.com/software/solutions/softwaremigration/dbmigteam.html>

## 6.2 Porting database server objects

The main steps required to convert a database and application include:

1. Porting preparation and installation
2. Database structure porting
3. Porting the database objects
4. Data porting
5. Converting database objects
6. Application conversion
7. Testing and tuning
8. User education

In this section, we focus on database object porting. Database objects, such as stored procedures, triggers, and user defined functions contain application logic which is stored in the database. We explain the syntax and implementation differences of the objects between MySQL and DB2. We provide examples to show you how to convert these objects from MySQL to DB2.

### 6.2.1 Stored procedures

A stored procedure is a set of SQL statements with flow logic that is stored and executed on the database server. Stored procedures help reduce network traffic since only the original request and the final output needed to be transmitted between the client and the server. As a server-side application, stored procedures can be accessed by client applications written in different languages on different platforms. Stored procedure support was added to MySQL in Version 5.1

#### ***MySQL stored procedures***

Similar to DB2, MySQL stored procedures follow the SQL2003 standard syntax. Similar to a programming language methods and functions, stored procedures have a name and parameter list, as well as SQL statements, error handling, and condition handling logic. Inside a stored procedure, you can also declare and return open cursors. MySQL stored procedures are created with the CREATE PROCEDURE statement. An application can invoke a stored procedure using the CALL statement. A MySQL stored procedure name is not case sensitive, but must be unique inside a database. Further, it must not be named the same as a system built-in function name. For example, now() cannot be used as a stored

```

>--+-----+-----*----->
'-(--+-----+--)-'
| .,-----|
| V .-IN---|
|'-----+-- parameter-name -- data-type +--'
|
|+-----+
|+OUT---+
|'-INOUT-'
|
|.-LANGUAGE SQL-.
>-- * --+----->
|
|.----+NO SQL-----+
|.----+MODIFIES SQL DATA-----+
>--*-----+----->
|.----+CONTAINS SQL-----+
|.----+READS SQL DATA-----+
|
|.-NOT DETERMINISTIC-.
>--*-----+-----*----->
'|-----DETERMINISTIC-----'

```

### Example 6-1 Syntax for creating SQL stored procedure in MySQL

```

>-+-----+--+*-----+----->
      .-SQL SECURITY-.      .- INVOKER-.

>--| SQL-procedure-body |-----><

```

---

### ***DB2 stored procedure***

DB2 stored procedures can be coded in the SQL PL language, or in compiled libraries using a third-generation language, including C, C++, COBOL, .NET common language runtime (CLR) languages, OLE, and Java. Stored procedures coded in a third generation language are known as *external* stored procedures. The focus of this section is on SQL PL stored procedures.

DB2 stored procedures can be coded in the SQL PL language, or in compiled libraries using a third-generation language, including C, C++, COBOL, .NET common language runtime (CLR) languages, OLE, and Java. Stored procedures coded in a third generation language are known as *external* stored procedures. The focus of this section is on SQL PL stored procedures.

The maximum number of supported parameters in a procedure is 32 767. Nested stored procedures are supported in SQL PL, Java, and C procedures, up to a limit of 16 nesting levels. Parameter default values are not supported, nor are

In DB2, once the CREATE PROCEDURE statement is executed, procedural and SQL statements in the procedure are converted into a native representation which is stored in the database catalogs. When an SQL procedure is called, the native representation is loaded from the catalogs and the DB2 engine executes the procedure.

### Example 6-2 Syntax for creating SQL stored procedure in DB2

## 358 Developing PHP Applications for IBM Data Servers



```

'- NO EXTERNAL ACTION -'

>--+-----+--+ * ----->
  '- PARAMETER CCSID --+ ASCII ----+'
    '- UNICODE -'

>--| SQL-procedure-body |-----><

```

---

## 6.2.2 Porting MySQL stored procedures to DB2

Converting MySQL stored procedures to DB2 is fairly straightforward since both MySQL and DB2 are based on the same SQL procedural language standard. Example 6-3 shows a sample MySQL stored procedure.

*Example 6-3 MySQL stored procedure*

---

```

DELIMITER $$
CREATE PROCEDURE `dlrshp`.`vehicle_purchase`(IN custid INTEGER, IN dlrid
INTEGER, IN vehid INTEGER, OUT odrid INTEGER)
BEGIN
  DECLARE discount INTEGER;
  DECLARE sales_price INTEGER;
  DECLARE vehicle_price INTEGER;
  SELECT SUM(p_amount) INTO discount
    FROM promotion
   WHERE f_v_id = vehid AND p_start_date <= NOW() AND p_end_date >= NOW();

  SELECT v_price INTO sales_price FROM vehicle WHERE v_id = vehid;
  SET vehicle_price = sales_price - discount;
  UPDATE inventory SET i_quantity = i_quantity - 1 WHERE f_v_id = vehid;
  INSERT INTO orders (f_c_id, f_d_id, f_v_id, o_indate, o_outdate, o_price)
  VALUES (custid, dlrid, vehid, current_date, current_date, vehicle_price);
  SELECT MAX(o_id) INTO odrid FROM orders;
END $$

```

---

Example 6-4 shows the corresponding DB2 stored procedure converted from the MySQL stored procedure in Example 6-3. There are not many changes required other than the differences in returning an open result set.

*Example 6-4 DB2 stored procedure*

---

```

CREATE PROCEDURE vehicle_purchase(IN custid INTEGER, IN dlrid INTEGER, IN
vehid INTEGER, OUT odrid INTEGER)
BEGIN
  DECLARE discount INTEGER;
  DECLARE sales_price INTEGER;
  DECLARE vehicle_price INTEGER;

```

```

DECLARE c1 CURSOR WITH RETURN FOR
SELECT * FROM customer;
SELECT SUM(p_amount) INTO discount
  FROM promotion WHERE f_v_id = vehid
  AND p_start_date <= CURRENT DATE AND p_end_date >= CURRENT DATE;
SELECT v_price INTO sales_price FROM vehicle WHERE v_id = vehid;
UPDATE inventory SET i_quantity = i_quantity - 1 WHERE f_v_id = vehid;
SET vehicle_price = sales_price - discount;
INSERT INTO orders (f_c_id, f_d_id, f_v_id, o_indate, o_price)
  VALUES (custid, dlrid, vehid, CURRENT DATE, vehicle_price);
SELECT MAX(o_id) INTO odrid FROM orders;
OPEN c1;
END@

```

---

We now show what has to be changed when porting MySQL stored procedures to DB2 by discussing the syntax and implementation differences.

## CREATE PROCEDURE

The CREATE PROCEDURE statement in both the databases are very similar.

MySQL:

```

CREATE PROCEDURE `dlrshp`.`vehicle_purchase`(IN custid INTEGER, IN dlrid
INTEGER, IN vehid INTEGER, OUT odrid INTEGER)

```

DB2:

```

CREATE PROCEDURE vehicle_purchase(IN custid INTEGER, IN dlrid INTEGER, IN
vehid INTEGER, OUT odrid INTEGER)

```

## BEGIN ... END

The equivalent of MySQL's BEGIN...END is BEGIN [NOT ATOMIC] ...END in DB2. There are no transactional save points established by BEGIN...END in MySQL. The following is a simple example of a MySQL procedure which uses:

BEGIN...END:

```

DELIMITER \
CREATE PROCEDURE proc_mysql()
BEGIN
---set of SQL's being executed
END \

```

When the above stored procedure is ported to DB2, you can use the optional NOT ATOMIC clause in the BEGIN...END declaration, as shown below.

```

CREATE PROCEDURE proc_db2(OUT variable INT)
BEGIN NOT ATOMIC
---set of SQL's being executed

```

END@

In DB2, the `BEGIN ATOMIC` statement indicates that if an unhandled exception condition occurs in the compound statement, all SQL statements in the compound statement will be rolled back.

## DECLARE

The `DECLARE` statement is used to declare local variables, conditions, cursors, and condition handlers. While porting MySQL procedure to DB2, changes are rarely required, except for potential data type differences.

In DB2, you also can declare return codes. You can use the `SIGNAL` and `RESIGNAL` statements to explicitly raise a custom error with a custom `SQLSTATE` or return a user-defined error message using the `SET MESSAGE_TEXT` statement.

## DELIMITER

A stored procedure requires two kinds of delimiters: one to denote the end of a statement or declaration within the procedure, and another to delimit the end of the procedure.

MySQL uses the `DELIMITER` clause to specify the character used to mark the end of a stored procedure. The commonly used delimiter in MySQL is `\`, for example:

```
DELIMITER \
```

In DB2, you must specify the delimiter you have chosen when creating the procedure from the command line. This is done by using the `td` option in the `db2` command. In the example below, the `@` is used to denote the end of a stored procedure in the `stored_procedure_file`.

```
db2 -td@ -vf <stored_procedure_file>
```

## SQL SECURITY

The concept of security is different in DB2 and MySQL. In DB2, user accounts exist in an external security facility, such as the operating system, while in MySQL, this may not be the case. The MySQL stored procedure `SQL SECURITY` clause is used to specify whose permission should be used when the stored procedure is invoked. The two choices are:

- ▶ **DEFINER**: The user who creates the stored procedure.
- ▶ **INVOKER**: The user who invokes or calls the stored procedure.

The default value is `DEFINER`.

In DB2, a user can create a stored procedure in DB2 database only if the user has:

- ▶ SYSADM or DBADM authority
- ▶ BINDADD privilege on the database, sufficient privileges to execute the SQL statements contained in the procedure, and either the IMPLICIT\_SCHEMA privilege or CREATEIN privileges

The stored procedure creator implicitly has EXECUTE WITH GRANT OPTION permissions on the stored procedure created. This user or the database administrator can grant execution permission to other users using GRANT EXECUTE command.

```
GRANT EXECUTE ON PROCEDURE myproc_db2 TO PUBLIC
```

The above command gives EXECUTE privilege on procedure myproc\_db2 to all users. Permissions can be granted to a specific group, user, or to public.

After porting a MySQL stored procedure to DB2, you only need to grant the execute privilege to the INVOKER. For more details about the authorities required for stored procedures and user defined functions, refer to DB2 Information Center:

<http://publib.boulder.ibm.com/infocenter/wsad512/index.jsp?topic=/com.ibm.etools.subbuilder.doc/topics/rauthorities.html>

## Returning a result set

In a MySQL stored procedure, a SELECT statement opens and returns a result set directly. In DB2, you need to declare a cursor using the WITH RETURN FOR clause and open it before exiting the procedure.

The following example shows a MySQL stored procedure which returns a single result set.

```
CREATE PROCEDURE myproc_mysql()  
BEGIN  
    SELECT c_id FROM customer;  
END \\\
```

The corresponding DB2 stored procedure is shown below:

```
CREATE PROCEDURE myproc_db2()  
BEGIN  
    DECLARE c1 CURSOR WITH RETURN FOR  
        SELECT c_id FROM customer;  
    OPEN c1;  
END @
```

## Variables

Variable SET statements in stored procedures are the same in both DB2 and MySQL.

### *Session variables*

MySQL supports session variables (variable names starting with @) inside stored procedures. The value of a session variable persists across a session. Session variables are typically used to hold the value of OUT parameters, as shown in the example below:

```
mysql> call dlrshp.vehicle_purchase(1,1,1,@out);
mysql> select @out
```

In the CALL statement above, the value of the @out session variable is set in the stored procedure. Its value can be displayed using the select @out statement.

You can use INOUT or OUT parameters to convert MySQL session variables to DB2. The following example shows a typical usage scenario of a session variable inside a MySQL stored procedure.

```
DELIMITER \\  
CREATE PROCEDURE session_mysql()  
BEGIN  
SET @s_variable = 100;  
END \\  

```

The converted DB2 stored procedure is shown below. The session variable is converted to an OUT parameter. The value of s\_variable can then be accessed after the stored procedure finishes execution.

```
CREATE PROCEDURE session_db2(OUT s_variable INT)  
BEGIN  
SET s_variable = 100;  
END@
```

## Conditions and handlers

Both EXIT and CONTINUE handlers are similar in MySQL and DB2. DB2 has an additional UNDO handler which is not supported by MySQL.

## Additional DB2 stored procedure features

DB2 stored procedures are more mature and function rich than MySQL. For example, the following features are supported in DB2 stored procedures:

- ▶ DB2 Development Center is an invaluable tool for creating and debugging SQL and Java stored procedures.
- ▶ The CALLED ON NULL INPUT clause allows you to invoke the stored procedure regardless of whether or not any input parameter is NULL.

- ▶ You can specify the encoding scheme to use for all string data passed into and out of the procedure using the `PARAMETER CCSID` clause.
- ▶ You can have multiple stored procedures with the same name but different signatures (that is, with a different number of parameters or with different parameter data types).
- ▶ You can specify save points inside stored procedures using the `SAVEPOINT LEVEL` clause in the `CREATE PROCEDURE` statement.

### 6.2.3 Triggers

A trigger is a database object containing application logic which is activated when a particular event/operation occurs on a database table. For example, you could set up a trigger that checks the credit rating for a customer and is triggered each time a new customer is inserted into the `CUSTOMER` table. Triggers are supported in MySQL starting in Version 5.02. DB2, therefore, provides a more mature implementation of triggers.

Unlike stored procedures, triggers are associated with a database *table*, an *operation* (`INSERT`, `UPDATE`, `DELETE`) on the table, and a *point in time* (`BEFORE`, `AFTER`) which all must be specified when creating the trigger.

Example 6-5 shows the MySQL `CREATE TRIGGER` syntax diagram:

*Example 6-5 MySQL CREATE TRIGGER syntax*

---

```
CREATE TRIGGER <trigger name>
  { BEFORE | AFTER } { INSERT | UPDATE | DELETE }
ON <table name>
FOR EACH ROW <triggered SQL statement>
```

---

DB2 triggers have more options, as can be seen from the DB2 `CREATE TRIGGER` statement syntax diagram in Example 6-6.

*Example 6-6 CREATE TRIGGER syntax for DB2*

---

```
CREATE TRIGGER-- trigger-name --+-NO CASCADE BEFORE-+----->
                                     +-AFTER-----+
                                     '-INSTEAD OF-----'
```

```
>--+-INSERT-----+--ON--+- table-name +------>
      +-DELETE-----+          '- view-name --'
      '-UPDATE-----+-'
           |
           | .-,-----, |
           |   V         |
           '-OF--- column-name --'-'
```

---



#### *Example 6-8 After INSERT trigger*

---

```
CREATE TRIGGER check_ins AFTER INSERT ON vehicle
REFERENCING NEW as N
FOR EACH ROW
BEGIN ATOMIC
INSERT INTO inserthistory VALUES (N.v_price, CURRENT TIMESTAMP);
END@
```

---

To retrieve the values, you can query the table.

### **Additional DB2 trigger features**

The following list of advanced DB2 trigger features can enhance your application:

- ▶ DB2 provides a facility to create triggers on a view. This is accomplished by specifying the `INSTEAD OF` clause in the `CREATE TRIGGER` statement.
- ▶ The granularity of a trigger in MySQL is for `FOR EACH ROW`. In DB2, you can additionally specify `FOR EACH STATEMENT`, which means that the triggered action is to be applied only after statement processing is complete.
- ▶ In DB2, both transition variables (`NEW`, `OLD`) and transition tables (`NEW_TABLE`, `OLD_TABLE`) can be specified, where MySQL only has transition variables.
- ▶ DB2 has `ATOMIC` compound SQL statements, which means that if an error occurs in the compound statement, all SQL statements in the compound statement will be rolled back, and any remaining SQL statements in the compound statement are not processed.

Example 6-9 shows a sample MySQL trigger which is activated *before* the `VEHICLE` table is updated.

#### *Example 6-9 MySQL trigger*

---

```
DELIMITER //
CREATE TRIGGER check_upd BEFORE UPDATE ON vehicle
FOR EACH ROW
BEGIN
IF NEW.v_price < 0 THEN
SET NEW.v_price = 0;
ELSEIF NEW.v_price > 100000 THEN
SET NEW.v_price = 100000;
END IF;
END//
```

---

Example 6-10 on page 367 shows the equivalent DB2 trigger. Note how you can further restrict when a DB2 trigger activates by specifying the `ON` clause. In



Example 6-10, the `check_upd` trigger will only fire when the `v_price` column of the `vehicle` table is updated. If the `ON` clause is omitted, the trigger will activate every time the `vehicle` table is updated.

*Example 6-10 DB2 trigger*

---

```
CREATE TRIGGER check_upd BEFORE UPDATE OF v_price ON vehicle
REFERENCING NEW as N
FOR EACH ROW
BEGIN ATOMIC
IF N.v_price < 0 THEN
SET N.v_price = 0;
ELSEIF N.v_price > 100000 THEN
SET N.v_price = 100000;
END IF;
END@
```

---

## 6.2.4 Views

Views are used to encapsulate table data and frequently run queries. MySQL supports the use of views starting in Version 5.1.

Example 6-11 shows the `CREATE VIEW` syntax diagram in MySQL.

*Example 6-11 MySQL CREATE VIEW syntax*

---

```
CREATE
[OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW <view-name> [(<column-names>)]
AS <select-statement>
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

---

Example 6-12 shows DB2's `CREATE VIEW` syntax diagram.

*Example 6-12 DB2 CREATE VIEW syntax*

---

```
-CREATE--VIEW-- view-name ----->

>--+-----+--AS----->
+-(---- column-name +--)-----+
'-OF-- type-name --+| root-view-definition |--'
               '-| subview-definition |---'

>--+-----+-- fullselect -- * ----->
|          .-,------. |
```



In DB2, both Materialized Query Tables (MQTs) and Views are made up of queries. In views, the query, on which the view is based, is executed every time the view is referenced. As opposed to MQTs, where the query results are materialized to disk and the data can be retrieved from the MQT directly instead of from the base tables.

### **OR REPLACE**

MySQL's OR REPLACE clause is not available in DB2. In DB2, if the view name to be created already exists, you must explicitly drop it first using the DROP VIEW statement.

### **SQL SECURITY**

MySQL uses the SQL SECURITY clause to determine which MySQL account to use when checking access privileges for the view when the view is executed.

The security mechanism in DB2 is different. To define a view, the person must have one of the following privileges:

- ▶ SYSADM or DBADM authority, or
- ▶ CONTROL or SELECT privilege for each table, view, or nickname identified in any full select and at least one of the following:
  - IMPLICIT\_SCHEMA authority on the database, if the implicit or explicit schema name of the view does not exist
  - CREATEIN privilege on the schema, if the schema name of the view refers to an existing schema

Once the view is created, you can use the GRANT statement to give other users privileges on the view. For example, the following statement gives the SELECT privilege to all users for the view db2inst1.view\_mergetb1.

```
GRANT SELECT ON db2inst1.view_mergetb1 TO PUBLIC
```

### **ALGORITHM = UNDEFINED/MERGE/EMPTYTABLE**

This MySQL syntax is not part of the SQL2003 standard. In general, if the MySQL view is defined with ALGORITHM = MERGE, the conversion is straightforward. Example 6-13 shows a MySQL view using the ALGORITHM=MERGE clause. When converting this view to DB2, you simply need to remove the SQL SECURITY, DEFINER, and ALGORITHM clauses. As previously noted, MySQL's NO REPLACE clause is converted to a DROP VIEW statement in DB2.

*Example 6-13 View in MySQL using MERGE algorithm*

---

```
CREATE OR REPLACE  
ALGORITHM=MERGE  
DEFINER=`root`@`localhost`
```

```

SQL SECURITY DEFINER
VIEW view temptbl
AS SELECT v_id, v_type, v_model, sum(i_quantity) as total
FROM inventory, vehicle
WHERE f_v_id = v_id group by v_id, v_type, v_model ;

```

---

*Example 6-14 DB2 conversion of MySQL view using MERGE algorithm*

---

```

DROP VIEW view mergetbl;
CREATE VIEW view mergetbl AS
SELECT v_id, v_type, v_model, sum(i_quantity) as total
FROM inventory, vehicle
WHERE f_v_id = v_id group by v_id, v_type, v_model;

```

---

In MySQL, when the `ALGORITHM = TEMPTABLE` clause is specified, a temporary table is materialized to hold the results of the view. This temporary table then is by the statement referencing the view. If the `ALGORITHM = TEMPTABLE` clause is used in conjunction with aggregate functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so on), `DISTINCT`, `GROUP BY`, `HAVING`, `UNION`, or `UNION ALL`, you can convert the MySQL view to a DB2 view directly.

Example 6-15 shows a MySQL view using the `TEMPTABLE` clause.  
 Example 6-16 shows the equivalent DB2 view.

*Example 6-15 View in MySQL using TEMPTABLE algorithm*

---

```

CREATE OR REPLACE
ALGORITHM=TEMPTABLE
DEFINER='root'@'localhost'
SQL SECURITY DEFINER
VIEW view temptbl
AS SELECT v_id, v_type, v_model, sum(i_quantity) FROM
inventory, vehicle WHERE f_v_id = v_id GROUP BY v_id, v_type, v_model;

```

---

*Example 6-16 DB2 conversion of MySQL view using TEMPTABLE algorithm*

---

```

DROP VIEW v_tmptble_db2;
CREATE VIEW v_tmptbl_db2 (car_id, car_type, car_model, car_quantity)
AS (
SELECT v_id, v_type, v_model, sum(i_quantity) FROM inventory, vehicle WHERE
f_v_id = v_id GROUP BY v_id,v_type,v_model);

```

---

The `ALGORITHM = UNDEFINE` clause is the default. In this case, MySQL will decide to either use `MERGE` or `TEMPTABLE`, since the algorithm is based on how the view can be resolved based on the view definition. Since MySQL views with using `MERGE` or `TEMPTABLE` algorithms can be converted to DB2 with

only minor syntax changes, the `ALGORITHM = UNDEFINE` clause can be converted in the same way.

### Additional DB2 view features

DB2 views are much mature, robust, and feature rich than MySQL views. Triggers defined against Views is a DB2 feature that MySQL does not support. These are known as `INSTEAD OF` triggers. Once you have converted your application to DB2, you can begin to take advantage of this functionality.

## 6.2.5 User Defined Functions

MySQL refers to user defined functions (UDFs) as *stored functions*. The MySQL stored function implementation is the same as stored procedures except that the stored function can only return a scalar value. A MySQL stored function can be invoked inside any SQL statement, such as any other built-in function. In DB2, SQL functions are used the same way, but provide more flexibility in that you can return a scalar value, a table, or a row.

The following example shows a very simple scalar function. This statement can be executed on both DB2 and MySQL without any changes.

```
CREATE FUNCTION inverse (X DOUBLE)
  RETURNS DOUBLE
  LANGUAGE SQL
  DETERMINISTIC
  RETURN (1)/(X);
```

Since the MySQL stored function implementation is the same as stored procedures, stored procedure properties and behaviors apply. This means the conversion process is the same.

## 6.3 Porting a client PHP application

This section is targeted to application developers who are planning to migrate their PHP applications to IBM DB2 UDB V8.2 from MySQL 5.

You will need to choose one of the following migration strategies based on what interface you are currently using to connect to MySQL 5. These options provide the least disruptive migration strategies, but there may be a better solution in the long term.

Table 6-1 shows four common migration options.

Table 6-1 MySQL to DB2 interface migration options

Current MySQL interface	Easiest DB2 interface
PDO	PDO
mysql	ibm_db2
mysqli	ibm_db2
PHP abstraction layer such as PEAR DB or ADOdb	Same abstraction layer, but a different connection string. There are caveats for using the Unified ODBC functions that these layers wrap. You should consider migrating to PDO, a native abstraction layer.

There are several interfaces for connecting to MySQL and IBM DB2 databases from PHP. If you are currently using PDO with MySQL, you are in luck, since all that will need to change is the string by which you establish a connection to the database. “Changing the connection string for PDO” on page 373 shows the lines that will need to change.

If you are connecting to MySQL 5 with the `mysql` or `mysqli` functions, “Function mapping for the `mysql` functions to `ibm_db2`” on page 374 shows how these functions map to the `ibm_db2` functions for working with IBM DB2.

6.3.1, “Interface” on page 372 demonstrates the new `ibm_db2` code that you will need to use to perform basic database interaction by showing how to connect, query, disconnect, and handle errors.

6.3.2, “Prepared statements” on page 384, 6.3.3, “Transactions” on page 386, and 6.3.4, “Stored procedures” on page 388 demonstrate how to migrate advanced database features such as prepared statements, transactions, and stored procedure code from the `mysqli` functions to `ibm_db2`.

**Tip:** If you are currently using a PHP abstraction layer, such as PEAR DB or ADOdb, you should consider replacing your code with the PDO native abstraction layer. PEAR DB and ADOdb wrap the Unified ODBC functions, which are no longer recommended.

## 6.3.1 Interface

As described in Chapter 4, “PHP application development with DB2” on page 95, there are three methods for interacting with a DB2 database from PHP.

- PDO for PHP 5.

- ▶ `ibm_db2` for PHP 4 and PHP 5.
- ▶ Unified ODBC for PHP 4 and PHP 5, though this is not recommended and therefore not covered.

In MySQL V5, you have three access interfaces:

- ▶ PDO for PHP 5
- ▶ `mysql` for PHP 4 and 5, which can connect to all MySQL versions, but only has limited functionality for working with V4.1 and V5.
- ▶ `mysqli` is only for PHP 5 and MySQL V4.1 and v5.

In the case of either `mysql` or the procedural method of using `mysqli`, it is relatively straightforward to replace those calls with the equivalent DB2\_ functions, though there are important differences.

## Changing the connection string for PDO

If you are already using PDO for MySQL 4.1, you will be able to just change the data source connection string. This is possible because PDO defines a common interface for interacting with a database, regardless of the vendor. You will still need to compile PHP to support the DB2 PDO\_IBM/PDO\_ODBC driver as described in 4.2.2, “PDO\_IBM/PDO\_ODBC” on page 139.

### *Example 6-17 MySQL connection in PDO*

---

```
$db = new PDO("mysql:host=$hostname;dbname=$database", $user, $pass);
```

---

The MySQL connection string in PDO as shown in Example 6-17 can be changed to one of the following DB2 connection strings, depending on whether you are connecting to a database that has already been cataloged on the client. Example 6-18 shows how to change the connection string for a cataloged database connection.

### *Example 6-18 Cataloged DB2 connection in PDO*

---

```
$db = new PDO("odbc:DSN=$database;UID=$user;PWD=$password;");
```

---

Example 6-19 shows the connection string for an uncataloged connection to a DB2 database.

```
$db = new PDO("odbc:DSN={IBM DB2 ODBC  
DRIVER};HOSTNAME=$hostname;PORT=$port;DATABASE=$database;PROTOCOL=TCPIP;UID=$us  
er;PWD=$password;", '', '');
```

### Function mapping for the mysql functions to ibm\_db2

If you have configured PHP with the `--with-mysql` flag and are using the `mysql` functions, the easiest migration path is to use the `ibm_db2` interface. You may also decide to perform a more comprehensive code rewrite to use PDO.

The `mysql` functions have been the traditional method for connecting to MySQL from PHP. Table 6-2 shows a selected mapping of the common procedural `mysql` functions to the procedural `ibm_db2` functions. Note that the `mysql` functions only provide a single interface for retrieving database errors, while the `ibm_db2` functions provide interfaces for handling both connection and statement errors.

Table 6-2 *mysql to ibm\_db2 function mapping*

MySQL function	ibm_db2 function
mysql_connect	db2_connect
mysql_pconnect	db2_pconnect
mysql_close	db2_close
mysql_query	db2_exec
	db2_prepare
	db2_execute
mysql_fetch_*	db2_fetch_*
mysql_errno	db2_conn_error
mysql_error	db2_conn_errormsg
mysql_error	db2_stmt_error
mysql_error	db2_stmt_errormsg

### Function mapping for the mysqli functions to ibm\_db2

If you have configured PHP with the `-with-mysqli` flag and are using the `mysqli` functions, the easiest migration path is to use the `ibm_db2` interface. You may also decide to perform a more comprehensive code rewrite to use PDO.



The `mysqli` functions are the improved method for accessing MySQL V4.1 and V5 database servers from PHP 5 code. These functions provide a way to access the new features of these servers. However, they cannot be used with earlier versions of MySQL or with PHP 4.

Table 6-3 shows a selected mapping of the common MySQLi functions to the `ibm_db2` functions.

*Table 6-3 MySQLi to ibm\_db2 function mapping*

MySQLi function	ibm_db2 function
<code>mysqli_connect</code>	<code>db2_connect</code>
<code>mysqli_pconnect</code>	<code>db2_pconnect</code>
<code>mysqli_close</code>	<code>db2_close</code>
<code>mysqli_query</code>	<code>db2_exec</code>
	<code>db2_prepare</code>
	<code>db2_execute</code>
<code>mysqli_fetch_*</code>	<code>db2_fetch_*</code>
<code>mysqli_connect_errno</code>	<code>db2_conn_error</code>
<code>mysqli_connect_error</code>	<code>db2_conn_errormsg</code>
<code>mysqli_stmt_errno</code>	<code>db2_stmt_error</code>
<code>mysqli_stmt_error</code>	<code>db2_stmt_errormsg</code>

## Connecting and disconnecting

You must change the places in your code where you connect to your database by replacing your calls to `mysql_connect`, `mysql_select_db`, `mysqli_connect`, and `mysqli_select_db` with the equivalent `ibm_db2` extension functions.

The connection string will be different depending on whether you have cataloged the database, but the handle returned will be the same.

Example 6-20 shows a PHP script that uses the `mysql` extension to connect to a MySQL database server and use the selected database.

*Example 6-20 Connecting to a MySQL database with `mysql`*

```
<?php
// Connection values.
$hostname = 'localhost';
$database = 'dlrshp';
```

```

$user      = 'root';
$password = '123456';

// Connect to MySQL with mysql functions.
$db = mysql_connect($hostname, $user, $password);

// Select the database.
mysql_select_db($database);

// Check to see if the connection worked.
if ($db) {
    echo 'Connection succeeded';
    // Close the connection.
    mysql_close($db);
}

?>

```

---

Example 6-20 on page 375 shows how a PHP script uses the `mysqli` extension to connect to a MySQL database server and use the selected database.

---

*Example 6-21 Connecting to a MySQL database with `mysqli`*

---

```

<?php
// Connection values.
$hostname = 'localhost';
$database = 'dlrshp';
$user      = 'root';
$password = '123456';

// Connect to MySQL with mysqli functions.
$db = mysqli_connect($hostname, $user, $password);

// Select the database.
mysqli_select_db($db, $database);

// Check to see if the connection worked.
if ($db) {
    echo 'Connection succeeded';
    // Close the connection.
    mysqli_close($db);
}

?>

```

---

Example 6-22 shows how you can use the `ibm_db2` extension to connect to and use a cataloged database.

*Example 6-22 Connecting to a cataloged DB2 database with ibm\_db2*

---

```
<?php
// Connection values.
$database = 'dlrshp';
$user      = 'db2inst1';
$password  = '123456';

// Connect to DB2 with ibm_db2 functions.
$db = db2_connect($database, $user, $password);

// Check to see if the connection worked.
if ($db) {
    echo 'Connection succeeded';
    // Close the connection.
    db2_close($db);
}
?>
```

---

Example 6-23 shows how to use the `ibm_db2` extension to connect to and use an uncataloged database.

*Example 6-23 Connecting to an uncataloged DB2 database with ibm\_db2*

---

```
<?php
// Connection values.
$database = 'dlrshp';
$user      = 'db2inst1';
$password  = '123456';
$hostname  = 'tungsten.itsosj.sanjose.ibm.com';
$port      = 50000;

// Construct a connection string.
$dsn = "DRIVER={IBM DB2 ODBC
DRIVER};DATABASE=$database;HOSTNAME=$hostname;PORT=$port;PROTOCOL=TCPIP;UID=$us
er;PWD=$password;";

// Connect to DB2 with ibm_db2 functions.
$db = db2_connect($dsn, '', '');

// Check to see if the connection worked.
if ($db) {
    echo 'Connection succeeded';
    // Close the connection.
    db2_close($db);
}
?>
```

---

## Querying

You must change the places in your code where you run SELECT queries against your database by replacing calls to `mysql_query`, `mysql_fetch_*`, `mysqli_query`, and `mysqli_fetch_*` with the equivalent `ibm_db2` functions.

Example 6-24 shows a PHP script which uses the `mysql` interface to query the selected database.

*Example 6-24 Querying a MySQL database with mysql*

---

```
<?php
// Connection values.
$hostname = 'localhost';
$database = 'dlrshp';
$user      = 'root';
$password  = '123456';

// Connect to MySQL with mysql functions.
$db = mysql_connect($hostname, $user, $password);

// Select the database.
mysql_select_db($database);

// Execute a query.
$result = mysql_query('SELECT * FROM VEHICLE');

// Iterate through the resultset.
while($row = mysql_fetch_assoc($result)) {
    echo $row['V_ID'];
    echo $row['V_TYPE'];
    echo $row['V_MODEL'];
}

// Close the connection.
mysql_free_result($result);

// Close the connection.
mysql_close($db);
?>
```

---

Example 6-25 shows a PHP script which uses the `mysqli` extension to connect to a target, a database, and issue a query.

*Example 6-25 Querying a MySQL database with mysqli*

---

```
<?php
// Connection values.
$hostname = 'localhost';
```

```

$database = 'dlrshp';
$user      = 'root';
$password  = '123456';

// Connect to MySQL with mysqli functions.
$db = mysqli_connect($hostname, $user, $password, $database);

// Execute a query.
$result = mysqli_query($db, 'SELECT * FROM VEHICLE');

// Iterate through the resultset.
while($row = mysqli_fetch_assoc($result)) {
    echo $row['V_ID'];
    echo $row['V_TYPE'];
    echo $row['V_MODEL'];
}

// Close the resultset.
mysqli_free_result($result);

// Close the connection.
mysqli_close($db);
?>

```

---

To convert these scripts to DB2, simply replace the calls to `mysqli_query` and `mysqli_fetch_*`, or `mysqli_query` and `mysqli_fetch_*`, with `db2_exec`, `db2_fetch_assoc`, and `db2_free_result` as shown in Example 6-26.

---

*Example 6-26 Querying a DB2 database using the `ibm_db2` extension*

---

```

<?php
// Connection values.
$database = 'dlrshp';
$user      = 'db2admin';
$password  = 'db2inst1';

// Connect to DB2 with ibm_db2 functions.
$db = db2_connect($database, $user, $password);

// Prepare the query.
$result = db2_exec($db, 'SELECT * FROM DB2INST1.VEHICLE');

// Fetch the results.
while ($row = db2_fetch_assoc($result)) {
    echo $row['V_ID'];
    echo $row['V_TYPE'];
    echo $row['V_MODEL'];
}

```

```
// Close the resultset.
db2_free_result($result);

// Close the connection.
db2_close($db);
?>
```

---

## Issuing updates

You must also change the places in your code where you update the database by replacing your calls to `mysql_query` or `mysqli_query` with the `ibm_db2` extension functions `db2_exec` and `db2_free_result`.

Example 6-27 is a PHP script which uses the `mysql` extension to update a database.

*Example 6-27 Updating a DB2 database with mysql*

---

```
<?php
// Connection values.
$hostname = 'localhost';
$database = 'dlrshp';
$user      = 'root';
$password  = '123456';

// Connect to MySQL with mysql functions.
$db = mysql_connect($hostname, $user, $password);

// Select the database.
mysql_select_db($database);

// Execute a query.
$result = mysql_query('UPDATE INVENTORY SET I_QUANTITY = I_QUANTITY - 1 WHERE
V_ID = 100');

// Verify the update succeeded.
if ($result) {
    echo 'Update succeeded.';
}

// Close the connection.
mysql_close($db);
?>
```

---

Example 6-29 on page 381 shows a PHP script which uses the `mysqli` extension to update a database.

#### *Example 6-28 Updating a DB2 database with mysqli*

---

```
<?php
// Connection values.
$hostname = 'localhost';
$database = 'dlrshp';
$user     = 'root';
$password = '123456';

// Connect to MySQL with mysqli functions.
$db = mysqli_connect($hostname, $user, $password, $database);

// Execute a query.
$result = mysqli_query($db, 'UPDATE INVENTORY SET I_QUANTITY = I_QUANTITY - 1
WHERE V_ID = 100');

// Verify the update succeeded.
if ($result) {
    echo 'Update succeeded.';
}

// Close the connection.
mysqli_close($db);
?>
```

---

Example 6-29 shows a converted PHP script which uses the `ibm_db2` extension to update a database.

#### *Example 6-29 Updating a DB2 database with ibm\_db2*

---

```
<?php
// Connection values.
$database = 'dlrshp';
$user     = 'db2admin';
$password = 'db2inst1';

// Connect to DB2 with ibm_db2 functions.
$db = db2_connect($database, $user, $password);

// Prepare the query.
$result = db2_exec($db, 'UPDATE DB2INST1.INVENTORY SET I_QUANTITY = I_QUANTITY
- 1 WHERE V_ID = 100');

// Verify the update succeeded.
if ($result) {
    echo 'Update succeeded.';
}

// Close the connection.
```

```
db2_close($db);  
?>
```

---

## Handling errors

You must change the places in your code where errors are handled by replacing calls to `mysql_connect_errno`, `mysql_connect_errmsg`, `mysql_stmt_errno`, `mysql_stmt_errmsg`, or `mysqli_connect_errno`, `mysqli_connect_errmsg`, `mysqli_stmt_errno`, `mysqli_stmt_errmsg` with the equivalent `ibm_db2` functions.

Example 6-30 shows a PHP script which uses the `mysql` extension to handle errors.

*Example 6-30 Handling connection and statement errors with `mysql`*

---

```
<?php  
// Connection values.  
$hostname = 'localhost';  
$database = 'dlrshp';  
$user      = 'root';  
$password = '123456';  
  
// Connect to MySQL with mysql functions.  
$db = mysql_connect($hostname, $user, $password);  
  
// Select the database.  
mysql_select_db($database);  
  
// Check to see if the connection worked.  
if ($db) {  
    echo 'Connection succeeded';  
  
    // Execute a query.  
    $result = mysql_query('SELECT * FROM VEHICLE');  
  
    if ($result) {  
        echo 'Query succeeded.';  
    } else {  
        echo 'Query failed';  
        // Get the error message.  
        echo 'Error code: ' . mysql_errno();  
        echo 'Error message: ' . mysql_error();  
    }  
  
    // Close the connection.  
    mysql_close($db);  
} else {  
    echo 'Connection failed';  
    // Get the error message.
```



```
        echo 'Error code: ' . mysql_errno();
        echo 'Error message: ' . mysql_error();
    }
    ?>
```

---

Example 6-31 shows a PHP script which uses the `mysqli` extension to handle errors.

*Example 6-31 Handling connection and statement errors with `mysqli`*

---

```
<?php
// Connection values.
$hostname = 'localhost';
$database = 'dlrshp';
$user     = 'root';
$password = '123456';

// Connect to MySQL with mysqli functions.
$db = mysqli_connect($hostname, $user, $password, $database);

// Check to see if the connection worked.
if ($db) {
    echo 'Connection succeeded';

    // Execute a query.
    $result = mysqli_query($db, 'SELECT * FROM VEHICLE');

    if ($result) {
        echo 'Query succeeded.';
    } else {
        echo 'Query failed';
        // Get the error message.
        echo 'Error code: ' . mysqli_stmt_errno($result);
        echo 'Error message: ' . mysqli_stmt_error($result);
    }

    // Close the connection.
    mysqli_close($db);
} else {
    echo 'Connection failed';
    // Get the error message.
    echo 'Error code: ' . mysqli_connect_errno();
    echo 'Error message: ' . mysqli_connect_error();
}
?>
```

---

Example 6-32 shows the converted PHP code which uses the `ibm_db2` extension to get the error number and message for both connections and statements.

*Example 6-32 Handling errors with the `ibm_db2` extension*

---

```
<?php
// Connection values.
$database = 'dlrshp';
$user      = 'db2admin';
$password  = 'db2inst1';

// Connect to DB2 with ibm_db2 functions.
$db = db2_connect($database, $user, $password);

// Check to see if the connection worked.
if ($db) {
    echo 'Connection succeeded';

    // Execute a query.
    $result = db2_exec($db, 'SELECT * FROM DB2INST1.VEHICLE');

    if ($result) {
        echo 'Query succeeded.';
    } else {
        echo 'Query failed';
        // Get the error message.
        echo 'Error code: ' . db2_stmt_error();
        echo 'Error message: ' . db2_stmt_errormsg();
    }

    // Close the connection.
    db2_close($db);
} else {
    echo 'Connection failed';
    // Get the error message.
    echo 'Error code: ' . db2_conn_error();
    echo 'Error message: ' . db2_conn_errormsg();
}
?>
```

---

### 6.3.2 Prepared statements

Using the `mysqli` extension, prepared statements are handled as shown in Example 6-33.

*Example 6-33 Handling a prepared statement with the mysqli extension*

---

```
<?php
// Connection values.
$hostname = 'localhost';
$database = 'dlrshp';
$user     = 'root';
$password = '123456';

// Connect to MySQL with mysqli functions.
$db = mysqli_connect($hostname, $user, $password, $database);

// Prepare the query.
$stmt = mysqli_prepare($db, 'SELECT * FROM VEHICLE WHERE V_COLOR = ?');

// Bind a parameter.
$color = 'Blue';
mysqli_stmt_bind_param($stmt, 's', $color);

// Execute the query.
$result = mysqli_stmt_execute($stmt);

// Fetch the results.
$vehicles = array();
while ($row = mysqli_fetch_assoc($result)) {
    $vehicle = new VehicleV0();
    $vehicle->id = $row['V_ID'];
    $vehicle->type = $row['V_TYPE'];
    $vehicle->model = $row['V_MODEL'];
    $vehicles[] = $vehicle;
}

// Close the statement.
mysqli_stmt_close($stmt);

// Close the connection.
mysqli_close($db);
?>
```

---

Example 6-34 shows the converted PHP code which uses the functions of the ibm\_db2 extension.

*Example 6-34 Handling a prepared statement with the ibm\_db2 extension*

---

```
<?php
// Connection values.
$database = 'dlrshp';
$user     = 'db2admin';
$password = 'db2inst1';
```

```

// Connect to DB2 with ibm_db2 functions.
$db = db2_connect($database, $user, $password);

// Prepare the query.
$stmt = db2_prepare($db, 'SELECT * FROM VEHICLE WHERE V_COLOR = ?');

// Bind a parameter.
$color = 'Blue';
db2_bind_param($stmt, 1, 'color', DB2_PARAM_IN);

// Execute the query.
$result = db2_execute($stmt);

// Fetch the results.
$vehicles = array();
while ($row = db2_fetch_assoc($stmt)) {
    $vehicle = new VehicleV0();
    $vehicle->id = $row['V_ID'];
    $vehicle->type = $row['V_TYPE'];
    $vehicle->model = $row['V_MODEL'];
    $vehicles[] = $vehicle;
}

// Close the connection.
db2_close($db);
?>

```

---

### 6.3.3 Transactions

Using the `mysqli` extension, transactions are handled as shown in Example 6-35.

*Example 6-35 Handling a transaction with the `mysqli` extension*

---

```

<?php
// Connection values.
$hostname = 'localhost';
$database = 'dlrshp';
$user     = 'root';
$password = '123456';

// Connect to MySQL with mysqli functions.
$db = mysqli_connect($hostname, $user, $password, $database);

// Prepare the queries.
$stmt1 = mysqli_prepare($db, 'UPDATE VEHICLE SET V_PRICE = ? WHERE V_ID = ?');
$stmt2 = mysqli_prepare($db, 'UPDATE INVENTORY SET I_QUANTITY = I_QUANTITY - 1
WHERE F_V_ID = ?');

```

```

// Bind the parameters.
$price = '24000';
$id = 100;

mysqli_stmt_bind_param($stmt1, 'sd', $price, $id);
mysqli_stmt_bind_param($stmt2, 'd', $id);

// Set autocommit off
mysqli_autocommit($db, false);

// Execute the query.
$result = mysqli_stmt_execute($stmt1) && mysqli_stmt_execute($stmt2);

if ($result) {
    // Commit the transaction.
    mysqli_commit($db);
} else {
    // Or roll it back.
    mysqli_rollback($db);
}

// Close the statements.
mysqli_stmt_close($stmt1);
mysqli_stmt_close($stmt2);

// Close the connection.
mysqli_close($db);
?>

```

---

Example 6-36 shows the converted PHP code which uses the equivalent functions of the `ibm_db2` extension.

---

*Example 6-36 Handling a DB2 transaction in `ibm_db2`*

---

```

<?php
// Connection values.
$database = 'dlrshp';
$user      = 'db2admin';
$password  = 'db2inst1';

// Connect to DB2 with ibm_db2 functions.
$db = db2_connect($database, $user, $password);

// Prepare the queries.
$stmt1 = db2_prepare($db, 'UPDATE DB2INST1.VEHICLE SET V_PRICE = ? WHERE V_ID = ?');
$stmt2 = db2_prepare($db, 'UPDATE DB2INST1.INVENTORY SET I_QUANTITY = I_QUANTITY - 1 WHERE F_V_ID = ?');

```

```

// Bind the parameters.
$price = '24000';
$id = 100;

db2_bind_param($stmt1, 1, 'price', DB2_PARAM_IN);
db2_bind_param($stmt1, 2, 'id', DB2_PARAM_IN);

db2_bind_param($stmt2, 1, 'id', DB2_PARAM_IN);

// Set autocommit off
db2_autocommit($db, DB2_AUTOCOMMIT_OFF);

// Execute the queries.
$result = db2_execute($stmt1) && db2_execute($stmt2);

if ($result) {
    // Commit the transaction.
    db2_commit($db);
} else {
    // Or roll it back.
    db2_rollback($db);
}

// Close the connection.
db2_close($db);
?>

```

---

### 6.3.4 Stored procedures

When migrating calls to stored procedures in PHP code from MySQL to DB2, you will need to be aware of a few differences between calling those defined in MySQL versus those defined in DB2.

► Calling the stored procedure:

The syntax in MySQL and DB2 is slightly different. In MySQL, the input parameters are specified by “?” and the output parameters are specified by a variable name with a prefix @. For example:

```
CALL vehicle_purchase(?, ?, ?, @odrid)
```

In DB2, all placeholders in the CALL statement are specified by “?”. For example:

```
CALL vehicle_purchase(?, ?, ?, ?).
```

► PREPARE the statement

The syntax of the function calls to prepare the statement in both MySQL and DB2 are the same. The only change required is the name of the function called. No other change in syntax is required.

MySQL: **mysqli\_prepare(\$db, \$sql)**

DB2: **db2\_prepare(\$db, \$sql)**

► Bind

In MySQL, all the parameters are bound in one step:

```
mysqli_stmt_bind_param($stmt, 'ddd', $custid, $custid, $vehid)
```

In DB2, you need to bind each parameter separately. This will simplify the result access later. The MySQL bind function above is converted to DB2 as:

```
db2_bind_param($stmt, 1, 'custid', DB2_PARAM_IN);  
db2_bind_param($stmt, 2, 'dlrid', DB2_PARAM_IN);  
db2_bind_param($stmt, 3, 'vehid', DB2_PARAM_IN);  
db2_bind_param($stmt, 4, 'odrid', DB2_PARAM_OUT);
```

► Execute the query

In MySQL, you need to call `mysqli_multi_query` multiple times to get the status of the query execution and to execute the query. For example:

```
$result = mysqli_multi_query($db, $stmt)  
$result = mysqli_multi_query($db, 'SELECT @odrid')
```

The first call tests if the query execution is successful. The second one is used to actually execute the query.

In DB2, you only need one function call, `db2_execute`:

```
$result = db2_execute($stmt)
```

The `$result` variable contains the status of the statement execution and the result set is ready to be retrieved if the query is executed successfully.

► Access the result set

The function call in MySQL is `mysqli_fetch_assoc`. In DB2, change it to `db2_fetch_assoc`. The other syntax is the same.

Example 6-37 shows a complete PHP script in MySQL. The converted DB2 script is shown in Example 6-38 on page 390.

*Example 6-37 Calling a stored procedure in MySQLi*

---

```
<?php  
// Connection values.  
$hostname = 'localhost';  
$database = 'dlrshp';  
$user     = 'root';  
$password = '123456';
```

```

// Connect to MySQL with mysqli functions.
$db = mysqli_connect($hostname, $user, $password, $database);

if ($db) {
    $custid = 1;
    $dlrid = 1;
    $vehid = 1;
    $result = mysqli_multi_query($db, "SET @custid = $custid");
    $result = mysqli_multi_query($db, "SET @dlrid = $dlrid");
    $result = mysqli_multi_query($db, "SET @vehid = $vehid");

    $result = mysqli_multi_query($db, 'CALL vehicle_purchase(@custid, @dlrid,
@vehid, @odrid)');
    if (!$result) {
        echo 'Execution failed';
        // Get the error message.
        echo 'Error code: ' . mysqli_stmt_errno($stmt);
        echo 'Error message: ' . mysqli_stmt_error($stmt);
    } else {
        if ($result = mysqli_multi_query($db, 'SELECT @odrid')) {
            if ($result = mysqli_use_result($db)) {
                while ($row = mysqli_fetch_row($result)) {
                    echo 'Order id: ' . $row[0];
                }
            }
            mysqli_free_result($result);
        }
    }
} else {
    echo 'Connection failed';
    // Get the error message.
    echo 'Error code: ' . mysqli_connect_errno($db);
    echo 'Error message: ' . mysqli_connect_error($db);
}
?>

```

---

**Example 6-38** *Calling a stored procedure in ibm\_db2*

---

```

<?php
// Connection values.
$database = 'dlrshp';
$user = 'db2admin';
$password = 'db2inst1';

// Connect to DB2 with ibm_db2 functions.
$db = db2_connect($database, $user, $password);

if ($db) {

```



```

$custid = 1;
$dldrid = 1;
$vehid = 1;
$dodrid = 0;
$sql = 'CALL DB2INST1.vehicle_purchase(?, ?, ?, ?)';
$stmt = db2_prepare($db, $sql);
if (!$stmt) {
    echo 'Preparation failed';
    // Get the error message.
    echo 'Error code: ' . db2_stmt_error();
    echo 'Error message: ' . db2_stmt_errormsg();
} else {
    db2_bind_param($stmt, 1, 'custid', DB2_PARAM_IN);
    db2_bind_param($stmt, 2, 'dldrid', DB2_PARAM_IN);
    db2_bind_param($stmt, 3, 'vehid', DB2_PARAM_IN);
    db2_bind_param($stmt, 4, 'dodrid', DB2_PARAM_OUT);

    $result = db2_execute($stmt);
    if (!$result) {
        echo 'Execution failed';
        // Get the error message.
        echo 'Error code: ' . db2_stmt_error();
        echo 'Error message: ' . db2_stmt_errormsg();
    }

    }

    echo 'The out variable is: ' . $dodrid . '<br />';
    echo 'This is the first result set<br />';
    if ($row = db2_fetch_assoc($stmt)) {
        print_r($row);
    }
} else {
    echo 'Connection failed';
    // Get the error message.
    echo 'Error code: ' . db2_conn_error();
    echo 'Error message: ' . db2_conn_errormsg();
}
?>

```

---



# An introduction to Service Data Objects for PHP<sup>1</sup>

Service Data Objects (SDOs) have been around in the Java technology world since November 2003. They are designed as a means of simplifying and unifying working with heterogeneous data sources. In February 2005, IBM and Zend announced a strategic partnership to collaborate on the development and support of the PHP environment. One aspect of this collaboration has been the definition and implementation of SDOs for PHP. This article gives an overview of SDOs and the motivations for using them in the PHP environment. A simple contact management scenario is used to illustrate key concepts.

---

<sup>1</sup> This article was originally published by developerWorks (<http://www-128.ibm.com/developerworks/opensource/library/os-sdphp/>) in July 2005. Reprint by Permission of authors.

## A.1 Introduction

Service Data Objects (SDOs) are designed to simplify and unify the way applications handle data. Using SDOs, application programmers can uniformly access and manipulate data from heterogeneous data sources, including relational databases, XML data sources, Web services, and other such enterprise information systems.

**Note:** SDO for PHP currently has support for relational and XML data sources, and a service provider interface to enable the implementation of support for others.

SDOs are based on the concept of disconnected data graphs. A data graph is a collection of data objects. Under the disconnected data graphs architecture, a client retrieves a data graph from a data source, changes the data graph, and can then apply the data graph updates back to the data source.

The task of connecting applications to data sources is performed by a Data Access Service (DAS) (see Figure A-1). Client applications can query a DAS and get a data graph in response, modify the data graph and send the updated data graph to a DAS to have the changes applied to the original data source. Clients may also use DASes to read from one data source and write to another -- for example, reading an XML RSS feed and writing the results to a relational database. This SDO/DAS architecture allows applications to deal principally with data graphs and data objects.

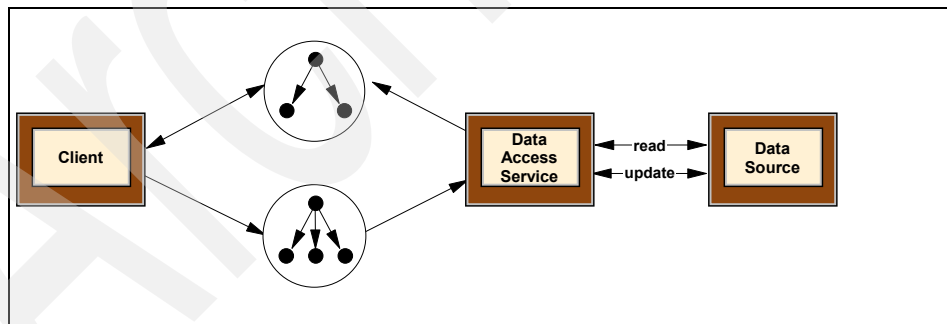


Figure A-1 The Role of a DAS

The PHP implementation of SDOs involves mapping to a dynamic and weakly typed language. The result is a greatly simplified API, with the collapsing of the many type-specific getters/setters. In addition to this, SDOs also exploit PHP's ability to implement objects that can be manipulated as if they were arrays,

including support for iteration, state testing and unsetting. The result is a powerful data object technology with a simple, intuitive interface.

## A.2 SDO concepts

Core SDO concepts are defined by the SDO model and a small number of interfaces that enable working with instance data managed in accordance with the model. Figure A-2 shows a UML class diagram of these concepts. Let's take a few moments to understand the roles of the different elements.

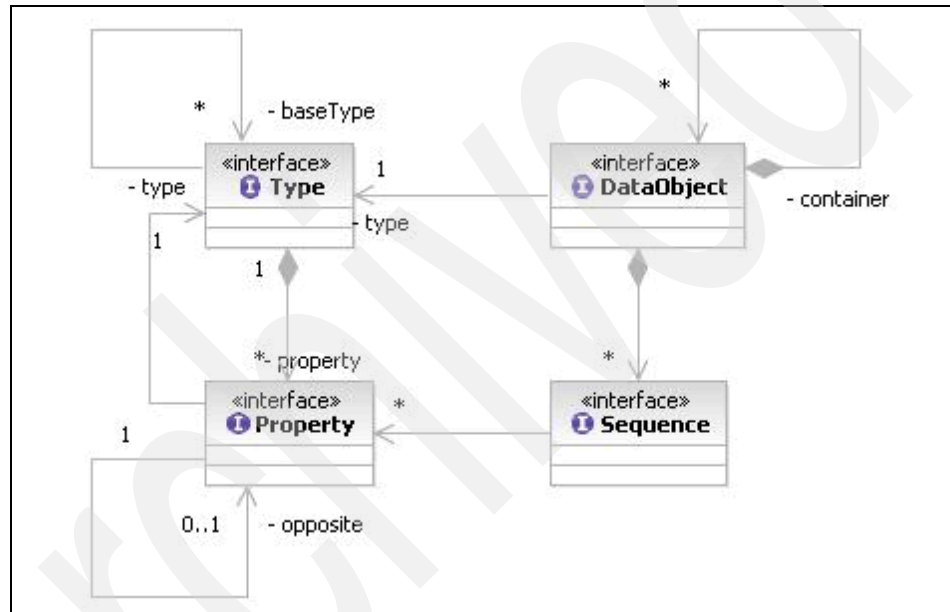


Figure A-2 SDO Model and Interfaces

### Type and Property

At the heart of every SDO instance is a model that defines the permitted structure of a data object. You can think of this as being like a blueprint for the data object. It covers concepts such as parent-child data object relationships, cardinality of relationships, permitted properties on a data object, default values. The SDO model is described in terms of *Types* and *Properties*. Types can be primitives, such as string (SDO calls these "data types" and defines a set mapped to PHP types), or *complex* types to represent an *order* or *address* (SDO calls these *data object types*). Data object types contain *properties*. Each property has a type that can be a data type or a data object type.

This Type and Property model allows data objects to represent what SDO refers to as *data graphs*. The essence of a data graph is a tree structure of data objects, navigable via their containment references (you can think of these as container, or aggregation, parent-child relationships), plus noncontainment references that point to data objects within the data graph (these are not aggregation relationships and can, therefore, point to any other data object within the tree).

Let's use the example of a Person data object to illustrate the important concepts of Type and Property. Figure A-3 shows an example of a Person SDO instance on the left and its corresponding model on the right. The Person SDO instance has its structure defined by a type with name of "Person." The Person type has been defined to have two properties with the names "name" and "age." The types of these two properties are SDO data types of "String" and "Int." On the left, we can see that the Person instance has had these two properties set to the values of "Fred Fish" and 35, respectively.

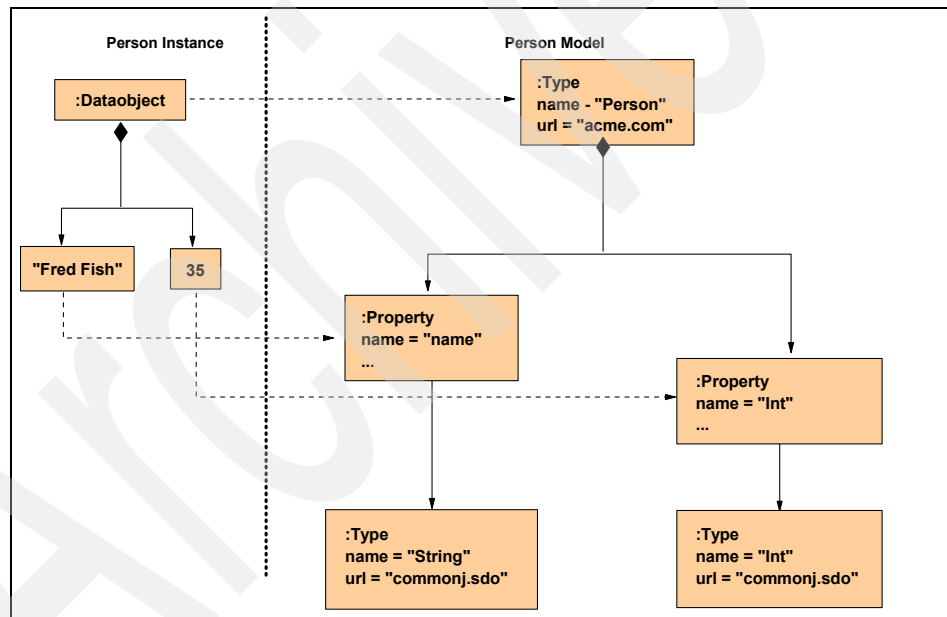


Figure A-3 Example Person SDO instance and model

In addition to the attributes mentioned, types also have attributes that say whether they are *open* (i.e., supports additional instance properties not defined in the type), *sequenced* (preserve order across properties), or *abstract* (an abstract base type to be extended by another SDO type). Properties also have attributes that describe things like cardinality (for example, a many-valued property to represent the departments in a company), any default value, whether the

property is read-only. Not all these concepts are supported in the PHP implementation of SDO. The documentation accompanying the SDO for PHP project describes the status of the various capabilities. All these concepts are covered in detail in the Java SDO specification. (see “Resource” on page 411).

**Note:** SDO for PHP does not currently support read-only properties, default values, abstract types, open types, or bidirectional relationships.

## DataObject

The core interface for working with data objects is the `DataObject` (no surprise there). This supports all the capabilities one would expect when working with a data structure, such as setting and getting properties, creating child structures, property querying via an augmented subset of XPath, and structure navigation.

If we consider the earlier example, we could use the `DataObject` interface to get the age property value, 35, and then use it to update the age to 36, say. The use of the `DataObject` interface is described in more detail in A.5, “Contact scenario” on page 400.

The sequence interface is used to manipulate data objects when ordering across a data object's properties is required (the data object is said to be a sequenced type). This is particularly useful when working with XML data where the order of property values is important, but varies for each instance and is, therefore, not explicitly defined by the model (for mixed XML content, for example).

## A.3 Data Access Services

As mentioned in the introduction, SDOs rely on the existence of DASes. The role of a DAS is to retrieve and write data to and from a data source. When using a DAS, the client works with SDOs, and is, therefore, insulated from any data source-specific data representation.

A DAS can also act as a factory for data objects, creating new instances that conform to some predefined data source schema (for example, a database or XML schema).

Two DAS implementations are provided as part of the SDO for PHP project. These are the XML DAS for working with XML files or XML/HTTP sources, and the Relational Data Access Service (Relational DAS), implemented using PHP Data Objects (PDO) for accessing relational data sources.

### A.3.1 Why choose Service Data Objects?

There are a number of reasons why you might consider using SDOs, the main ones outlined below.

#### **Reduced database overhead:**

At the heart of SDOs is support for disconnected working. Data objects can automatically record their change history that can then be used by a DAS to detect collisions when applying changes back to an enterprise information system.

This technique is often referred to as *optimistic concurrency*, or *optimistic offline locking* (described in *Patterns of Enterprise Application Architecture*, by Martin Fowler). It is best suited to scenarios where there is a low risk of collision, perhaps due to infrequent edits, or the edits being governed by some external process that reduces the likelihood of multiple people concurrently working on the same data.

The benefits are even greater if, in addition to the low risk of collision, there is also high edit latency (significant time between checking out to make edits and edits being committed), or the data is passed around in ways that make it difficult to manage database connections and locks (for example, in service-oriented application architectures).

The major benefit to using this technique is the removal of the need for the application to hold connections and locks in the enterprise information system while some user or application is working on the data.

Interestingly, there is nothing in the SDO architecture that precludes the creation of DASes that employ a pessimistic concurrency model.

#### **Single API for data**

In addition to the optimistic concurrency support, another major benefit to using SDO is realized when working with multiple heterogeneous data sources. SDOs provide a single API for manipulating data independent of the data's originating data source.

In addition to the expected data structure manipulation support, SDOs also provide the ability to set and retrieve the contents of an SDO based on an XPath-like expression, including queries. This removes the burden from the client of having to navigate the data structure in order to identify substructures based on instance data. For example, we might choose to extend our Person example to include a new data object type that contains a collection of Person data objects. To identify an individual by name without XPath would require us to iterate through all the Person objects and test the name property until a match



was found. To do this with XPath requires just a single call, specifying the property name and the value to be matched.

### Knowledge of the structure

As we have already seen, SDOs carry internal knowledge of the structure of the data they represent (the model). This knowledge is used to ensure that the creation and modification of instance data conforms to the structure and type rules for the data object.

Data capabilities, such as SimpleXML and PDO, do not employ this approach and, therefore, must delegate this responsibility to other technologies when creating and validating data. As a consequence, the benefit that SimpleXML and PDO have over SDO is they do not require the developer to specify the model.

An additional benefit to having a model is the capability to introspect it at run time. For example, this can enable developers to write flexible user interfaces that can adapt to changes in their data structures (SDO schema) at run time. SDO model introspection is not fully enabled in the PHP implementation, but we expect this to change over time.

**Note:** SDO for PHP currently only has limited support for model introspection (can determine the type of a data object, but not its structure), but we envisage this being improved over time

## A.4 Relationship to PHP Data Objects and SimpleXML

PDO (not to be confused with SDO) aims to provide a consistent API for the common capabilities found in most relational database APIs. This greatly simplifies creating Web applications designed to support different database vendors by encapsulating the differences under a common API. PDO provides a simple object view of results, but does not attempt to normalize those results (one row of the result set equals one object, regardless of whether there are multiple tables represented in the result). The ease of use of PDO makes it a natural choice when working directly with databases, and for this very reason is the technology chosen for the Relational DAS implementation provided with SDO.

As mentioned, the focus of SDO is on providing a flexible data object representation for data from heterogeneous data sources and built-in support for optimistic concurrency. We have also described how having a "knowledge of the structure" enables SDOs to provide a single API for the complete life cycle of the data, including creation and validation. If these capabilities are important to your

application, then SDO is probably an appropriate choice, using the Relational DAS implemented to PDO for relational data source support.

SimpleXML provides a simple way for working with XML instance documents. Documents are loaded, and can be navigated and manipulated through a simple API. The interface surfaces some specifics of XML (for example, the syntax differentiates between elements and attributes).

When parsing and processing an instance document, SimpleXML is an excellent technology choice. However, if significant manipulation is required, it is important that the technology understands the model for the document, and in that case, SDO is probably a more appropriate choice.

## A.5 Contact scenario

You should now have a reasonable understanding of the main concepts of SDO. To help clarify, we will illustrate them with a simple scenario.

The following sections provide an overview of SDO for PHP capabilities, described in the context of a personal contacts example. This example demonstrates the following attributes of SDO:

- ▶ Disconnected working (optimistic concurrency)
- ▶ SDO navigation and manipulation
- ▶ The role of a DAS

Code snippets are provided to help explain the concepts and development requirements. However, it is not our goal to describe a complete working example. We expect future articles to cover working examples and elaborate on the use of the relational and XML DASes.

### A.5.1 Contact edit use case

In this scenario, a contacts Web application has been written to support the management of contact information. The contact information is stored in a relational database containing the two tables as shown in Table A-1 and Table A-2 on page 401:

*Table A-1 "contact" table definition*

Column	Type	Example
shortname (primary key)	string	"Charlie"
fullname	string	"Charles Babbage"

Table A-2 "address" table definition

Column	Type	Example
id (auto-generated primary key)	integer	1
shortname (foreign key)	string	"Charles "
addressline1	string	"Analytical House"
addressline2	string	"1 Engine Close"
city	string	"Walworth Road"
state	string	"London"
zip	string	"XX11 1ZZ"
telephone	string	555-555-5555

To illustrate the use of SDOs, we have selected the use case of modifying a contact. The main steps to modifying a contact are as follows:

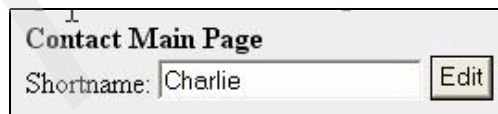
- ▶ Retrieve the contact to be modified from the database
- ▶ Make the modifications to the contact
- ▶ Apply the modification back to the database

These steps are described in more detail below.

**Note:** As we have mentioned, DAS APIs are not specified as part of SDO because we are referring to the SDO specification, and, therefore, any sample code is necessarily specific to a particular implementation. The code snippets shown below have been created to match the APIs of the Relational DAS provided with SDO for PHP.

## A.5.2 Retrieving the contact entry

The first page the user sees is the contact management main.php page. It has a single input field for entering the shortname of the contact to be edited, along with an Edit button to submit the request.



The screenshot shows a web form titled "Contact Main Page". Below the title is a label "Shortname:" followed by a text input field containing the text "Charlie". To the right of the input field is a button labeled "Edit".

Figure A-4 The contact management main page

Entering a shortname and clicking Edit transitions to the edit.php page, passing the specified shortname in the \$\_POST array. The edit.php page contains the following main steps:

1. Get the entered shortname
2. Create a relational DAS instance
3. Execute the query to retrieve the contact SDO
4. Populate the edit form with the contact details
5. Store the contact SDO in the session

Each of these steps is described in more detail below.

### 1. Get the shortname:

The welcome page form configuration resulted in the shortname being placed in \$\_POST['shortname'].

```
// get the shortname from posted variables
$shortname = $_POST['shortname'];
```

**Note:** At this point, and also in the section "Modifying the data," we would normally validate the user input to prevent the database from being compromised.

### 2. Create a Relational DAS instance

An important aspect of creating the Relational DAS is describing the database schema it should use. An example is shown in Example A-1. The Relational DAS will take this schema and use it to define the model for the Service Data Objects it can create. This information is often required by other Relational DAS instances and is, therefore, a candidate for placing in a separate script and then included.

*Example: A-1 Describing the database schema*

---

```
// Describe the structure of the contact table
$contact_table = array(
    'name' => 'contact',
    'columns' => array('shortname', 'fullname', 'telephone'),
    'PK' => 'shortname'
);

// Describe the structure of the address table
$address_table = array (
    'name' => 'address',
    'columns' => array('id', 'contact_id', 'addressline1', 'addressline2',
        'city', 'state', 'zip'),
    'PK' => 'id',
    'FK' => array ('from' => 'contact_id', 'to' => 'contact')
);
```

```

$table_metadata = array($contact_table, $address_table);

// Describe the parent-child relationship. This is information required
// by the Relational DAS to help map from the relational
// database representation to
// the data graph representation of SDO.
$address_reference = array('parent'
=> 'contact', 'child' => 'address');

```

---

**Note:** The Relational DAS assumes that all containment relationships are cardinality one-to-many. So in this example, the contact can contain zero or more address DataObject instances.

Having defined the model, we can now create an instance of the Relational DAS. A sample code is shown in Example A-2.

*Example: A-2 Creating a Relational DAS instance*

---

```

// Create the Relational Data Access Service telling it the database
// schema, that table should be considered the root of the graph,
// and finally the additional information for the object model.
$das = new SDO_DAS_Relational($table_metadata, 'contact',
$address_reference);

```

---

3. Execute the query:

We can now use the Relational DAS to retrieve the contact information. The PHP script is shown in Example A-3.

*Example: A-3 Use the Relational DAS to retrieve contact information*

---

```

// connect to the database. This connection will be released when the
// $dbh variable is cleaned up.
$dbh = new PDO('odbc:contactdb', DB_USER, DB_PASSWORD);
// construct the SQL query for contact retrieval
$stmt = "select * from contact, address where contact.shortname=$shortname
and
contact.shortname=address.contact_id";

// execute the query to retrieve the contact
$contact = $das->executeQuery($dbh, $stmt);

```

---

The resulting `$contact` SDO is shown in Figure A-5 on page 404. This shows the data objects, their properties (property index shown in the square brackets, followed by the property name), and the property values. As mentioned, the cardinality of the address containment within a contact is assumed by the Relational DAS to be one-to-many. In the diagram below, the

notation "[0] DataObject" has been used to signify the first entry in the list of address DataObjects.

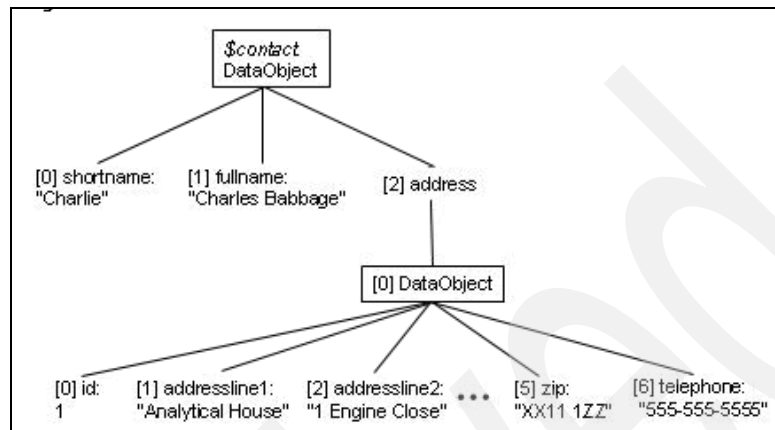


Figure A-5 Contact SDO instance

#### 4. Populate the edit form:

Given the contact SDO, we can populate the form to allow the user to edit the data.

*Example: A-4 Populate the edit form*

---

```

// Create and populate the form with the contact details
<form action= ... method="POST">
  <input type="text" name="fullname"
    value="$contact->fullname">
  ...
  <input type="text"
    "name="addressline1" value="
    $contact->address[0]->addressline1">
  ...
</form>

```

---

#### 5. Store the data object in the session:

Because we are disconnected from the database, we need to store the contact data object in the session to make it available to the next page.

```

// store the contact data object in the session
$_SESSION['contact_sdo'] = $contact;

```

### A.5.3 More on SDO navigation

The previous section briefly touched on accessing the properties of an SDO (see the code in Step 4, "Populate the edit form"). In addition to accessing primitive properties, most SDO applications also require navigation up and down parent-child data object relationships (between contact and address, for example). Some also require query capabilities to identify parts of the data graph.

The code snippets below give a quick overview of the ways of navigating the contact SDO data graph.

As we saw in Step 4, SDOs supports property access using the object property syntax:

```
// get the fullname using the object property
$fullname = $contact->fullname;
```

We can also access the fullname using the property index (the position as defined by the data object's model, as shown in the square brackets in Figure 4):

```
// get the second contact property (fullname)
$fullname = $contact[1];
```

We can access many-valued child data object properties, such as address, using the same syntax, see Example A-5.

*Example: A-5 Access many-valued child data object properties*

---

```
// get the list of address data objects via the object property
$addresses = $contact->address;

// get the list of address data objects via the property index
$addresses = $contact[2];
```

---

We can access individual elements of many-valued properties, such as the first address, using array syntax:

```
// get the first address from a list of address data objects
$address1 = $contact->address[0];
```

We can also directly reference properties within child data object properties, such as the ZIP code from the first address, see Example A-6 .

*Example: A-6 Reference properties within child data object properties*

---

```
// access the zip code from the first address via the object properties
$zip = $contact->address[0]->zip;

// access the zip code via the property indices.
```

```
// Note: this style is not recommended since it leads to virtually
// unserviceable code. The XPath-style (described later) or defining
// constants lead to much more readable code.
$zip = $contact[2][0][5];
```

---

We can also iterate over the properties of a data object as shown in Example A-7:

*Example: A-7 Iterate over the properties of a data object*

---

```
// Iterate over the properties of the first address
// $name is assigned each property name (e.g. "id", "addressline1", ...)
// $value is assigned each property value (e.g. 1, "Analytical House", ...)
foreach ($contact->address[0] as $name => $value) {
    echo "$name $value";
}
```

---

Finally, we can access the properties using XPath-like support, the simplest form being the property name, see Example A-8. :

*Example: A-8 Access the properties using XPath-like support*

---

```
// use property names (XPath) to access the zip property
$zip = $contact['address'][0]['zip'];

// use single XPath expression with array index notation
to access the zip property
// Note: XPath array indices start at 1.
$zip = $contact['address[1]/zip'];

// use single XPath expression with dotted index notation
to access the zip property
// Note: SDO dotted notation indices start at 0.
$zip = $contact['address.0/zip'];
```

---

XPath can also be used to navigate and query data objects. If we had retrieved a number of contacts in the Relational DAS query, we could identify an individual from its first address line; for example: The Relational DAS returns multiple results as a many-valued child property of a root data object, that in the example we have named \$root.

*Example: A-9 Use XPath to navigate and query data objects*

---

```
// Get the address object that contains the addressline1 of "1 Engine Close"
$address = $root["contact/address[addressline1='1 Engine Close']"];

// Get the contact with the matching address. GetContainer() navigates to the
// parent of a data object, in this case the contact SDO.
```



```
$contact = $address->getContainer();
```

---

## A.5.4 Modifying the data

Figure A-6 shows a simple example of a contact edit page, edit.php, where the contact has a single address.



Figure A-6 The contact edit page

This page allows the user to modify individual property values. When the Update button is clicked, all the values are placed in the `$_POST` array, regardless of whether they have been modified, and the application transitions to the confirm.php page. The confirm page performs the following main steps:

1. Retrieves the contact SDO from the session
2. Updates the contact SDO
3. Creates a Relational DAS instance
4. Writes the changes back to the database
5. Informs the user of the outcome

Each of these steps is described in more detail below.

### 1. Retrieve the contact SDO from the session:

The final step in the execution of the edit page was to place the contact SDO into the session. We now retrieve this contact to make the updates.

```
// retrieve the contact from the session
```

```
$contact = $_SESSION['contact_sdo'];
```

## 2. Update the contact:

Now that we have the contact, we can go about making the updates posted from the edit page. This is done by comparing the posted value with the old value, and if they are different, setting the posted value on the contact. We do this to avoid setting a value unnecessarily and causing SDO to record a change in the change summary (holds the old values for data objects that have been modified). It would be nice if SDO implementation were to do this test on our behalf. Example A-10 shows the sample code.

*Example: A-10 Update the contact*

---

```
// update the fullname if changed
if ($contact->fullname != $_POST['fullname']) {
    $contact->fullname = $_POST['fullname'];
}
...
```

---

## 3. Create a Relational DAS:

The next step is to create the Relational DAS used to write the updates to the database. This code is identical to that used in retrieval and, as mentioned, is best placed in a separate script, ("contact\_model.inc.php," for example).

```
// initialize the Relational Data Access Service
require_once('contact_model.inc.php');
```

## 4. Write the changes back to the database:

The next step is to apply the changes back to the database. The call below shows how this is done for the Relational DAS. There is no need to specify an SQL statement for updates because the Relational DAS derives this from the model and the contact data object's change summary.

```
// apply the changes back to the database
$das->applyChanges($dbh, $contact);
```

The `applyChanges()` call is deceptively simple. Under the covers, it is:

- Ordering SDO updates to ensure the correct results (for example, creates before updates).
- Generating SQL INSERT, UPDATE, and DELETE statements to apply the changes. The UPDATE and DELETE statements are qualified with the original values of the data so that should the data have changed in the database in the meantime this will be detected.
- Executing the SQL statements -- If any of the SQL statements fails to execute, this is an indication that a collision has occurred and the Relational DAS rolls back all changes and throws an exception. If all statements succeed, all the changes are committed to the database. The

client application can then continue to work with the data object, make more changes, and apply them, or can discard it.

**Note:** There are two common schemes employed for detecting conflicts:

- Add a version column (might be based on a timestamp) to each table, updated each time a row is modified. Comparing versions tells us if there is a conflict.
- Record all the original values and compare with the current ones to see if any have been modified.

The Relational DAS implements the second scheme since this does not require the database to be modified in order to use it.

## 5. Inform the user of the outcome:

The final task is to notify the user of the outcome. If no collisions are detected by the Relational DAS, and the update is successful, all is well.



**Contact Update Confirmation**  
Successfully updated: Charlie

Figure A-7 The confirmation page

## A.5.5 More on SDO modification

The previous section briefly touched on accessing and setting an SDO property (see the code in step 2 on page 408 “Update the contact:” on page 408). In addition to setting primitive properties, most SDO applications also require the creation of child data objects and the deletion of parts of the data structure. The code snippets below give a quick overview of the other types of modification one might wish to perform on the contact SDO.

The techniques described for getting individual properties are also available for setting. Example A-11 shows the sample code.

*Example: A-11 Setting individual properties*

```
// set the fullname via the object property
$contact->fullname = 'Alan Turing';

// set the fullname via the property index
$contact[1] = 'Alan Turing';

// set the fullname via the property name (XPath)
```

```
$contact['fullname'] = 'Alan Turing';
```

---

We can create child data objects. For example, the edit user interface could allow adding a new address to a contact. When new address details were posted, we might perform the actions as shown in Example A-12.

*Example: A-12 Create child data objects*

---

```
// create a child address data object
$address = $contact->createDataObject('address');

// set the address's addressline1 property from the posted value
$address->addressline1 = $_POST['addressline1'];
```

---

Note: This child data object is automatically inserted into the graph and `$address` is simply a reference to that position in the graph. So for example, if this were the first address added, the code shown in Example A-13 would both set the ZIP code on the contact's address.

*Example: A-13 Set ZIP code*

---

```
// set the address's zip
$address->zip = 'XY11 2ZZ';

// set the addresses zip via the contact data object
$contact->address[0]->zip = 'XY11 2ZZ';
```

---

We can test and unset individual instance properties of the contact. For example, if the user cleared the string in the interface, we could use this to signify unsetting, see Example A-14.

*Example: A-14 Test and unset individual instance properties*

---

```
// if the fullname value was cleared on the edit page and the fullname
// was previously set then unset the fullname property
if (empty($_POST['fullname']) && isset($contact->fullname)) {
    unset($contact->fullname);
}
```

---

Finally, we probably want to enable the deletion of a contact, or contact's address in the edit page, again implemented using unset, see Example A-15.

*Example: A-15 Enable deletion of a contact*

---

```
// test and unset the first address
if (isset($contact->address[0])) {
    unset($contact->address[0]);
}
```

}

---

## A.6 Summary

SDOs add some interesting capabilities for working with data in PHP, while maintaining the simple, easy-to-use interfaces PHP developers expect. SDOs can represent complex data structures from heterogeneous data sources, while allowing their manipulation through a single API similar to that of SimpleXML and PDO. Optimistic concurrency support is built into SDO, allowing disconnected data manipulation without requiring the application to implement change tracking and conflict detection.

This article has given a taste of some of the capabilities of SDO, but there are many that have not been covered. We expect subsequent articles to cover these topics, including:

- ▶ Different classes (or types) of data objects: sequenced, open, abstract
- ▶ Relational DAS details
- ▶ XML DAS details
- ▶ Implementing a DAS using SDO service provider interface

### Resource

#### *Learn*

- ▶ The SDO for PHP implementation is delivered as a *PECL* extension, and can be downloaded from the SDO project page at:  
<http://pecl.php.net/package/sdo>
- ▶ The SDO for PHP documentation can be found in the "Function Reference" section of the latest builds of the PHP documentation. The PHP documentation can be found at:  
<http://docs.php.net/>
- ▶ IBM and BEA are collaborating on specifications for programming models and APIs for Java 2 Enterprise Edition (J2EE™) application servers that provide programmers with simpler and more powerful ways of building portable server applications. Read about it in "*Service Data Objects, WorkManager, and Timers*" (developerWorks, June 2005). This article can be found at:  
<http://www.ibm.com/developerworks/library/specification/j-commonj-sdowmt/index.html>
- ▶ "Connecting PHP to Apache Derby" shows you how to install and configure PHP on Windows. This article can be found at:

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0409casey/>

- For extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM products, visit the developerWorks Open source zone at:

<http://www-128.ibm.com/developerworks/opensource>

### ***Get products and technologies***

Innovate your next open source development project with *IBM trial software*, available on DVD or for download at Web site.

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

### ***Discuss***

Get involved in the developerWorks community by participating in *developerWorks blogs* at:

<http://www.ibm.com/developerworks/blogs/>

## Additional material

This redbook refers to additional material that you can download from the Internet as described below.

### Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG247218>

Alternatively, you can go to the IBM Redbooks Web site at:

[ibm.com/redbooks](http://ibm.com/redbooks)

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG24-7218.

### Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
<b>sg24-7218Sample.zip</b>	Zipped Code Sample

## System requirements for downloading the Web material

We recommend the following system configuration

<b>Hard disk space:</b>	500 MB minimum
<b>Operating System:</b>	Windows 2000
<b>Processor:</b>	Pentium
<b>Memory:</b>	512 MB

## How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.



# Related publications

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

## IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 417. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *MySQL to DB2 UDB Conversion Guide*, SG24-7093

## Other publications

These publications are also relevant as further information sources:

### IBM - DB2

- ▶ IBM DB2 UDB documentation: *What's New V8*, SC09-4848-01
- ▶ IBM DB2 UDB documentation: *Administration Guide: Implementation V8*, SC09-4820-01
- ▶ IBM DB2 UDB documentation: *Administration Guide: Performance V8*, SC09-4821-01
- ▶ IBM DB2 UDB documentation: *Administration Guide: Planning V8*, SC09-4822-01
- ▶ IBM DB2 UDB documentation: *Application Development Guide: Building and Running Applications V8*, SC09-4825-01
- ▶ IBM DB2 UDB documentation: *Application Development Guide: Programming Client Applications V8*, SC09-4826-01
- ▶ IBM DB2 UDB documentation: *Application Development Guide: Programming Server Applications V8*, SC09-4827-01
- ▶ IBM DB2 UDB documentation: *Call Level Interface Guide and Reference, Volume 1, V8*, SC09-4849-01
- ▶ IBM DB2 UDB documentation: *Call Level Interface Guide and Reference, Volume 2, V8*, SC09-4850-01

- ▶ IBM DB2 UDB documentation: *Command Reference V8*, SC09-4828-01
- ▶ IBM DB2 UDB documentation: *Data Movement Utilities Guide and Reference V8*, SC09-4830-01
- ▶ IBM DB2 UDB documentation: *Data Recovery and High Availability Guide and Reference V8*, SC09-4831-01
- ▶ IBM DB2 UDB documentation: *Guide to GUI Tools for Administration and Development*, SC09-4851-01
- ▶ IBM DB2 UDB documentation: *Installation and Configuration Supplement V8*, GC09-4837-01
- ▶ IBM DB2 UDB documentation: *Quick Beginnings for DB2 Clients V8*, GC09-4832-01
- ▶ IBM DB2 UDB documentation: *Quick Beginnings for DB2 Servers V8*, GC09-4836-01
- ▶ IBM DB2 UDB documentation: *Replication and Event Publishing Guide and Reference*, SC18-7568
- ▶ IBM DB2 UDB documentation: *SQL Reference, Volume 1, V8*, SC09-4844-01
- ▶ IBM DB2 UDB documentation: *SQL Reference, Volume 2, V8*, SC09-4845-01
- ▶ IBM DB2 UDB documentation: *System Monitor Guide and Reference V8*, SC09-4847-01
- ▶ IBM DB2 UDB documentation: *Data Warehouse Center Application Integration Guide Version 8 Release 1*, SC27-1124-01-01
- ▶ IBM DB2 UDB documentation: *DB2 XML Extender Administration and Programming Guide Version 8 Release 1*, SC27-1234-01
- ▶ IBM DB2 UDB documentation: *Federated Systems PIC Guide Version 8 Release 1*, GC27-1224-01
- ▶ IBM Recovery Expert for Multiplatforms: *User's Guide*, SC18-9564-01
- ▶ *DB2 Grouper, V1R1, User's Guide*, SC18-7409-06

#### **IBM - Cloudscape**

- ▶ *IBM Cloudscape Server and Administration Guide Version 5.1*, SC18-9251

#### **IBM - Informix**

- ▶ *IBM Informix 4GL to EGL Conversion Utility User's Guide Version 6.0.0.1*, G251-2485
- ▶ *IBM Informix Database Design and Implementation Guide Version 10.0*, G251-2271
- ▶ *IBM Informix ESQL/C Programmer's Manual Version 9.53*, G251-1342

- ▶ *IBM Informix Guide to SQL: Reference Version 10.0*, G251-2283
- ▶ *IBM Informix Guide to SQL: Syntax Versions 10.0 and 8.5*, G251-2284

## Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ Cloudscape Tools and Utilities Guide Version 5.1, found at:  
<http://www.ibm.com/software/data/Cloudscape/version51/pubs/>
- ▶ DB2 Information Center  
<http://publib.boulder.ibm.com/infocenter/db2help/index.jsp>
- ▶ IBM Informix Dynamic Server information center  
<http://publib.boulder.ibm.com/infocenter/ids9help/index.jspDescription2>

## How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)



# Index

## Symbols

++ 131  
.htaccess 229  
.NET 8, 10, 131, 357  
.php 407  
.rhosts 229  
? 245  
@ 145  
@ operator 316, 321

## Numerics

32-bit 5  
64-bit 5, 190

## A

abstract 396, 411  
abstract type 397  
account 78  
add promotion interface 49–50  
add review form 44  
administration server 98  
ADO 8  
affectedRows() 238  
aggregate function 370  
aggregation 396  
ALARMPROGRAM 195  
ANSI 332  
ANSI logging 326  
apache2ctl 200, 217  
application xiv, 353–354, 356, 358, 366  
arameter marker 125  
array 168, 394  
assessment 355  
AUTOCOMMIT 115, 117–118, 147  
autocommit mode 326

## B

base table 332  
beans 61  
bidirectional relationship 397  
bindParam 153–154  
bindValue 153

bison 97  
BLOB 88, 304  
bottleneck 346  
Breakpoints 90  
buffer pool 176, 346

## C

C 95, 131, 357  
C# 8  
C++ 170, 357  
call stack 90  
case sensitiv 356  
catalog 100, 102, 184–185  
certified xiv  
checkpoint 346  
child structure 397  
chown 192  
CLI trace 170–171  
client 356, 358, 371  
CLOB 88  
CLP 358  
CLR 8, 357  
COBOL 357  
collection 394  
collision 398  
column 52–53, 61, 122, 129, 135–138, 149–151, 157–158, 167–169  
command line environment 99  
Command Line Processor 358  
committed read 331  
common API 399  
common language runtime 357  
communication 99, 170  
community 412  
complex data structure 411  
complex type 395  
complexity 355  
connect call 231  
connection 384  
connection pooling 228  
connection resource 114, 120–121, 139, 161–162  
console 80  
constraints 10

- contact detail 36
- contact information 403
- Control Center 98, 176, 178–181, 183
- controller 32, 56–59, 65
- conversion 353–356, 369–371
- cost based optimizer 11
- COUNT() 370
- cpp 97
- CPU 333
- CREATE PROCEDURE 132
- critical error 319
- CS 119–120
- CURRENT ISOLATION 151
- cursor 118–121, 138, 148, 152, 159, 161, 174, 254
- cursor stability 331
- cursorstability 119
- customer registration 36–37

## D

- DAS 394, 403, 407, 411
- Data 354–358, 364, 369, 373
- Data Access Service 394
- data caching 10
- data flow 234
- data graph 394
- data graphs architecture 394
- data model 32–33, 53, 56, 60
- data objec 398
- data object 397
- Data replication 6
- data source 394
- Data Source Name 141–142
- data type 122–124, 127, 135, 152–153, 156
- data warehousing environment 6
- database adapter 67, 70
- database design xiv
- database solution 5
- databases server 5
- DataObject 397, 404
- DB2 Express-C 7
- DB2 family 5
- DB2\_ATTR\_CASE 137
- db2\_autocommit 117–118
- DB2\_AUTOCOMMIT\_OFF 117
- DB2\_AUTOCOMMIT\_ON 117
- DB2\_BINARY 114, 123, 130
- db2\_bind 125
- db2\_bind\_param 122–124, 129, 133, 173

- DB2\_CASE\_LOWER 137
- DB2\_CASE\_NATURAL 137
- DB2\_CASE\_UPPER 137
- DB2\_CHAR 123
- db2\_close 114, 116–118, 139
- db2\_connect 83, 114–117, 139, 173
- db2\_exec 379–380
- db2\_execute 123–124, 174
- db2\_fetch\_array 125, 136, 169
- db2\_fetch\_assoc 125, 137, 379
- db2\_fetch\_both 125, 138
- db2\_fetch\_object 125, 138, 174
- db2\_fetch\_row 125, 138
- db2\_field\_ 135
- db2\_free\_result 139, 379–380
- db2\_free\_stmt 139
- DB2\_LONG 123–124
- db2\_next\_result 133
- db2\_num\_records 169
- DB2\_PARAM\_IN 123–124, 133
- db2\_pconnect 114–118, 139
- db2\_prepare 122–123, 173
- db2\_rollback 118
- db2\_stmt\_error 119, 122
- db2\_stmt\_errormsg 119
- db2profile 99, 105, 108, 112
- db2setup.err 99
- db2setup.his 98
- dbschema 342
- DBSERVERNAME 195, 198
- dbspaces 253
- deadlock detection 10
- dealer inventory interface 46–47
- debugger 75, 89–91
- default value 98, 161, 397
- deletion 410
- delimiter 361
- design xiv
- developerWorks 412
- development environment 202
- development tool 97
- directory 78, 99, 101, 105, 107–108, 110, 144, 170–171, 174
- dirty read 331
- distribution 204
- domain object 60
- DRDA 99, 101
- DSN 141–142, 144, 163, 173
- dynamic SQL 71, 232

## E

e-business 6  
Edit mode 88  
EM64T 190  
enterprise information system 398  
environment 5  
environment variable 101, 113, 192, 195, 198,  
210–211, 213, 219, 221, 226, 349  
error message 321  
errorCode() 146, 316, 318  
errorInfo() 316, 318  
ESQL/C 308  
Event analyzer 179  
event monitor 180  
Execute 101, 114–115, 118, 121–124, 141, 148,  
151–153, 156, 162, 164–165, 171, 183  
extension 28, 59, 70, 102–105, 107–110, 140, 145,  
148, 151, 161, 165, 169, 174  
external table 333

## F

features 31–32, 38–40, 55, 71  
fenced user 97  
FETCH\_ASSOC 284  
fetchAll 250  
fetchRow 263  
file system 12  
flex 97  
forward only cursors 119  
functions 356, 358, 363, 371–374, 377

## G

gcc 97  
getMessage() 146, 322  
getters 394  
getUserInfo() 322  
group 78  
group commit 10  
groupadd 190  
gunzip 97, 192

## H

HADR 6  
heterogeneous 398, 411  
Hex Viewer 88  
High availability disaster recovery 6  
HTML 356

HTML table 259  
HTTP 2, 12  
httpd.conf 98, 105, 108–109  
HyperText Transfer Protocol 12

## I

IBM\_DB2 70, 103–104, 106, 110–111, 113, 118,  
120–121, 133, 136, 144, 148, 151, 161–162, 165,  
169, 174, 183  
IDE 83  
IFX PEAR 70  
ifx.byteasvarchar 308  
ifx.textasvarchar 308  
ifx\_ 70  
ifx\_\* 204, 245  
ifx\_connc 230  
ifx\_error 321  
ifx\_errormsg 321  
ifx\_fetch\_row 259  
ifx\_fetch\_row() 310  
ifx\_getsqlca() 236  
IFX\_HOLD 260  
ifx\_pconnect 230  
ifx\_query 327  
IHS 17  
IIS 79–80, 84  
Image Viewer 88  
IN parameter 131–132, 156  
INFORMIXSERVER 195, 198  
infrastructure 2  
ini\_set() 237  
INOUT parameter 132, 156  
installation 355–356  
instance 78–79, 358  
instance configuration file 194  
instance owner 97, 99–100  
intangible attribute 33  
integrity 3  
interface 3, 394  
Internal Debugger 89  
Internet 12  
internet service provider 353  
ipcs 98  
iSeries 6  
isolation level 10, 118–120, 148, 151, 165, 184,  
332  
ISP 353  
ISV 7

## J

J2EE 411  
Java 8, 357, 393  
Java virtual machine 10  
joins 11  
JVM 10

## K

kernel parameter 100  
keyset-driven scrollable cursor 119  
ksh 101  
-ky 315

## L

large object 129–130, 161  
LDAP 22  
libgcc 97  
libxml2 97  
libxml2-devel 97  
Lightweight Directory Access Protocol 22  
LIST 269  
lock mode 332, 343  
lock wait 332  
locking 10  
log in interface 38, 46  
logging database 329  
logic 4  
LOGSIZE 194  
LRU\_MAXDIRTY 347  
LRU\_MINDIRTY 347

## M

make 96–97, 99–100, 103, 105, 107, 111, 113, 119, 140–141, 147–148, 162–163, 171, 183, 185  
manipulation 400  
MAX() 370  
MB 96, 182  
memory setting 334  
MERGE statement 333  
message queue 98  
method 102, 106, 116, 125, 129, 140, 142–143, 146, 148, 151–152, 157, 161, 183  
midrange 6  
MIN() 370  
mkdir 192  
model 399  
model-view-controller 32

monitor 90  
MPM 15  
MSGPATH 195  
Multi-Processing Module 15  
multi-threaded 10  
multi-user 10  
MVC 32, 56–57, 61  
mysql\_connect\_errmsg 382  
mysql\_connect\_errno 382  
mysql\_fetch\_ 379  
mysql\_query 380  
mysql\_stmt\_errmsg 382  
mysql\_stmt\_errno 382  
mysqli\_functions 71  
mysqli\_connect\_errmsg 382  
mysqli\_connect\_errno 382  
mysqli\_fetch\_ 379  
mysqli\_query 379–380  
mysqli\_stmt\_errmsg 382  
mysqli\_stmt\_errno 382

## N

named parameter marker 153  
nameless parameter marker 153, 155  
navigate 406  
navigation 400  
network 3  
network Server 101–102  
nobody 98  
nogroup 98  
non persistent connection 114, 163  
non-functional requirement 33

## O

object relational database server 8  
object-oriented 31–32, 70–71  
ODBC 8, 71, 103–104, 107, 110, 113, 116, 139, 141–142, 144, 161, 163, 165, 169–170  
odbc\_close 169  
odbc\_connect 163  
odbc\_do 165–166, 169  
odbc\_exec 166, 303, 329  
odbc\_execute 165–166, 246, 303  
odbc\_fetch\_into 168–169  
odbc\_fetch\_object 168  
odbc\_fetch\_row 265  
odbc\_free\_result 169  
odbc\_pconnect 163



- odbc\_prepare 303
- odbc\_result 167, 265
- odbcinst.ini 226, 231
- onconfig 193
- oninit 196, 315
- online 2
- onsoctcp 193
- onstat 343
- open type 397
- optimistic concurrency 398, 400
- optimistic offline locking 398
- optimization 341
- optimizer hint 334
- OS/400 6
- OUT parameter 132
- outcome 33, 35–36, 38, 40–47, 49–51

## P

- package 96, 101, 103, 106
- page 407
- parameter 98–99, 115, 117, 121–124, 128, 130–131, 133, 143, 152–158, 161, 165, 169–170, 173, 175, 190, 195
- parameter array 125
- parameter marker 122–123, 153–155
- parent-child 396
- password 78, 88
- PDO
  - ATTR\_ERRMODE 102, 104, 107, 109, 139–142, 144–145, 147–151, 153–154, 157, 183
  - ERRMODE\_EXCEPTION 140–141, 145–148, 150, 152–153, 159
  - ERRMODE\_SILENT 104, 140–145, 147–153, 158
  - ERRMODE\_WARNING 140–142, 145, 147–150, 152–153
  - errorCode() 141, 145–147, 152–153, 156
  - errorInfo() 141, 145–147, 152–153, 156
- PDO\_ODBC 23, 71, 102–104, 109–110, 113, 139–141, 144, 147, 150, 153, 156
- PEAR DB 70–71, 212
- PEAR module, 204
- PEARL 9
- Performance 106, 114, 120–122, 140, 161, 169, 175, 177, 183
- permissions 276

- persisted information 4
- persistent connection 114–116, 118, 144, 161, 163, 183, 228, 231
- PHP 8
- PHP 4 70
- PHP 5 31–32, 56, 70
- PHP build configuraiton parameters
  - with-config-file-path 215
- PHP build configuration parameters
  - with-apxs2 107
  - with-pdo-odbc 107
  - with-pdo-odbc=ibm-db2 104
- PHP build confiruation parameters
  - with-IBM\_DB2 104
- PHP Data Objects 23
- php.ini 105, 113, 140, 144, 161
- physical log 347
- PHYSSIZE 194
- place holder 245
- port number 99, 101, 143, 184
- Porting 354–356, 359–362, 365, 368
- preconditions 35–36, 38, 40–47, 49–51
- Prepare 114, 121, 125, 141, 148, 151–153, 163, 165, 183
- primitive properties 405
- programming interface 5
- programming language 28
- programming language independent 2
- properties 395
- property 403
- Protocol 2
- protocol 99, 116, 142
- publishing function 127

## R

- RDBMS 5
- read atability 119
- readline-devel 97
- read-only 397
- Redbooks Web site 417
  - Contact us xvi
- Relational DAS 397
- relational database management systems 5
- reliability 8, 28
- repeatable read 119, 331
- requirements 31, 33, 35–38, 40, 42–43, 45–47, 49–53, 56
- retrieve 397

- RHEL4 204
- role 334
- root class 64
- ROOTPATH 194
- ROOTSIZE 194
- row type 270
- ROWNUMBER 138
- RR 119–120
- RS 119–120
- RSS 394
- runtime 100–102
- runtime client 78

## S

- SBLOB 315
- Scalable 3
- scroll cursor 253, 255
- Scrollable cursors 119, 134, 148, 159
- SDO 393–394, 407, 411
- SDO schema 399
- search vehicles interface 39
- Security 97, 122
- security 122, 131, 146
- self-build libraries 204
- sequence interface 397
- sequenced 396, 411
- sequential scan 345
- server 354–356, 375–376
- SERVERNUM 195
- Service Data Objects 393
- service port 99
- SET EXPLAIN ON 285
- setFetchMode 248, 261
- setters 394
- shared library 209
- shared lock 331
- shared nothing architecture 333
- shortname 401
- SimpleXML 411
- source code 96, 101, 103
- special register 151
- SQL 354–364, 366–367, 369, 371
- SQL PL 357
- SQL Query Function 88
- SQL Server Configuration 87
- SQL Server Tree 87
- SQLCODE 115, 146, 165
- SQLConnect 173

- sqlhosts.std 193
- SQLJ 8
- SQLSTATE 115–119, 146, 149–150, 165
- square bracket 403
- stability 28
- stability isolation 119–120
- standard cursor 255, 258
- state testing 395
- statement 384
- statement resource 114, 122, 125, 135–136, 161–162
- static method 59
- static SQL 232
- statistics 345
- Step in 90
- Step over 90
- stored procedure 97, 122, 131, 133, 156, 161, 169, 353, 356–361, 363, 372
- SUM() 370
- Sybase 354–355
- synchronization 6
- syntax 400
- sysdbclose 335
- sysdbopen 334
- SYSTEM user 98

## T

- table 41–42, 54–55, 61
- tar 97, 103, 106
- Task Center 98
- TCP/IP 99–100
- technology 399
- Text Viewer 88
- throughput 347
- timestamp 176, 409
- track order interface 43
- tracking activities 334
- transaction 325
- transfer object 61
- triggers 10, 353, 356, 358, 364–365
- Type and Property model 396
- types 395

## U

- UDR 9
- Uncommitted Read 119
- Unified ODBC 71
- unit of work 176

unixODBC 231, 312, 330  
untrusted session 230  
UR 119–120  
URL 12  
use case 33, 36–47, 49, 51  
user case 40  
user defined function 97  
user defined routines 9  
user type 229  
useradd 190  
utility 180

## **V**

version column 409

## **W**

WML 3  
workbench 11  
workload 6  
workshop 355  
write ahead logging 10

## **X**

XA transaction 10  
Xeon 190  
XML 125, 128, 184  
XMLSerialize 125  
XPath 397, 399  
XPath-like expression 398  
XPS 333

## **Z**

z/OS 5  
Zend Core for IBM 75–79, 81, 83  
Zend Studio 75, 83–85, 87, 89–91  
zero administration 11  
zlib-devel 97  
zSeries 190





## Developing PHP Applications for IBM Data Servers

(0.5" spine)  
0.475" <-> 0.875"  
250 <-> 459 pages







**Redbooks**

# Developing PHP Applications for IBM Data Servers

**Develop and deploy Web solutions using PHP and IBM data servers**

**See PHP with DB2, Informix IDS, and Cloudscape examples**

**Port PHP applications from MySQL to DB2**

PHP is one of the world's most popular programming languages for building dynamic data-driven Web applications, and IBM data servers store a majority of the world's data -- combining these two technologies is a natural step for any Web developer.

This IBM Redbook provides lots of information for developers, including code samples for creating PHP applications with DB2, Informix Dynamic Server (IDS), and Cloudscape. We use the latest PHP data access extensions including: PHP Data Objects (PDO) and the native `ibm_db2` extension for PHP.

We describe the installation and configuration details for setting up the IBM data servers and Apache Web application server for PHP applications. We include Zend Core for IBM, Zend Studio installation, and configuration.

In addition, we discuss the process of porting PHP applications from MySQL 5 to DB2 UDB V8.2.

## INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

### BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)