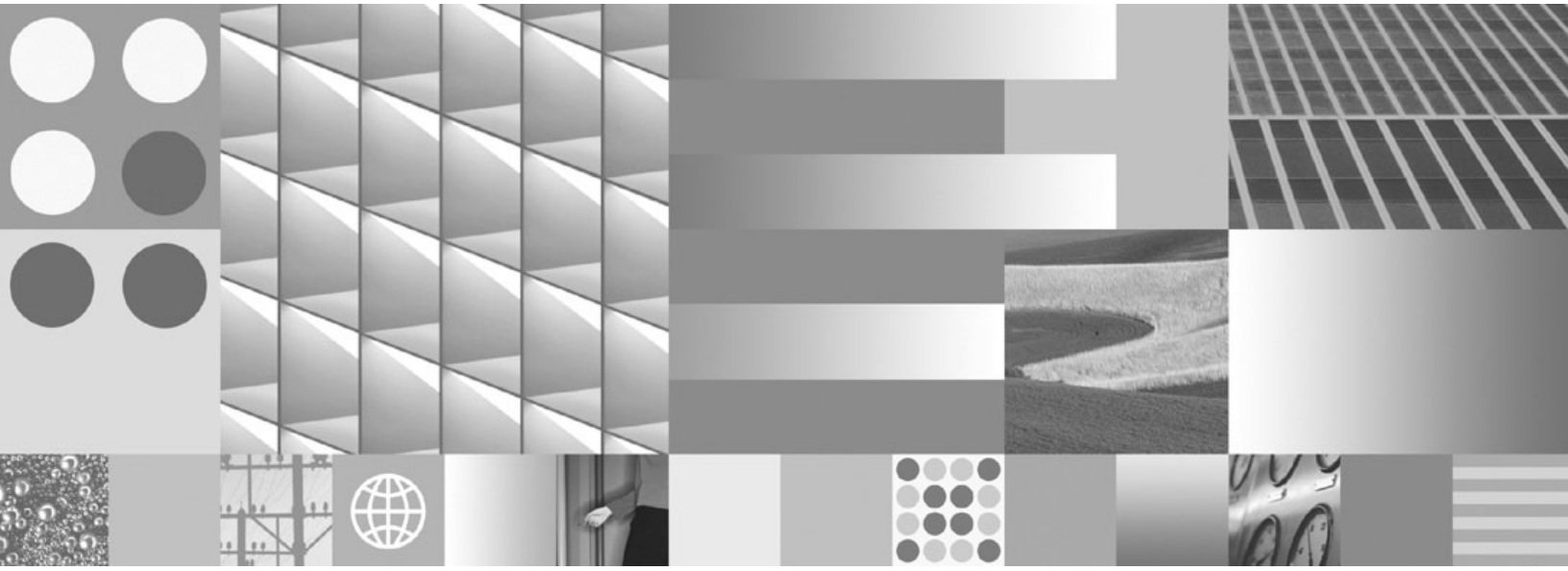
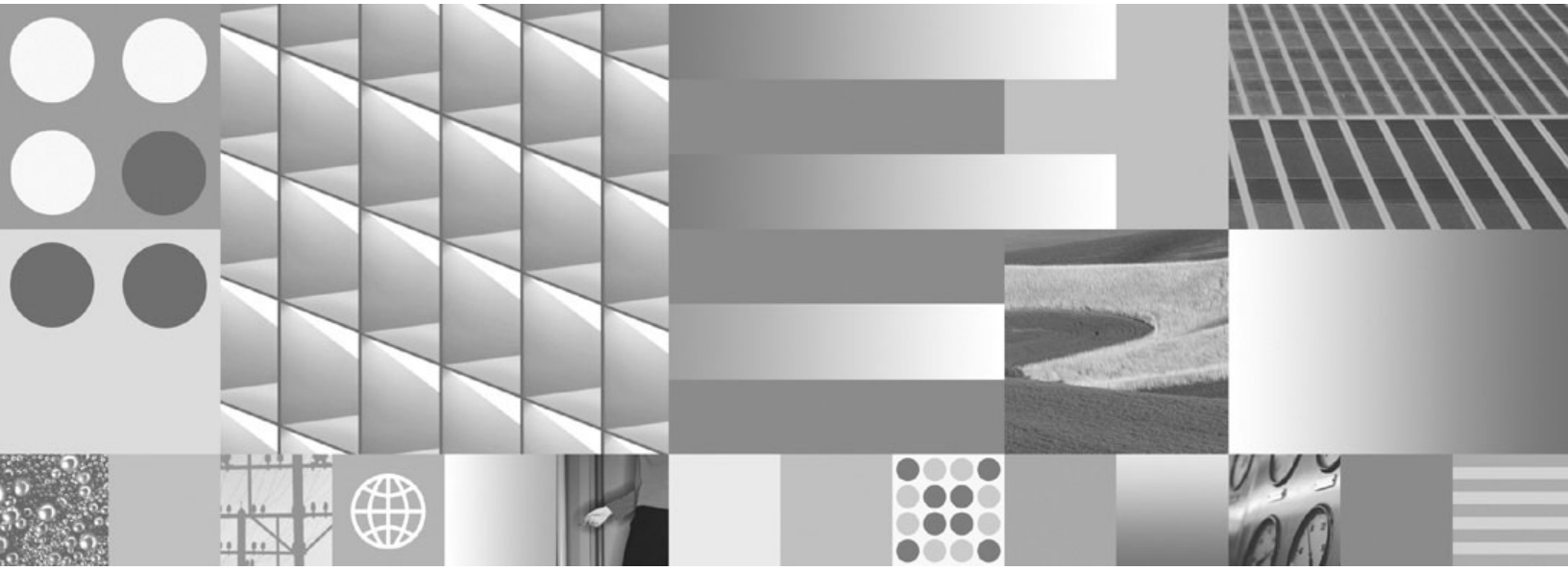
DB2 Version 9.5
for Linux, UNIX, and Windows



Developing Perl and PHP Applications

SC23-5854-00

DB2 Version 9.5
for Linux, UNIX, and Windows



Developing Perl and PHP Applications

SC23-5854-00

Note
Before using this information and the product it supports, read the general information under Appendix B, "Notices," on page 49.

Edition Notice

You can order IBM publications online or through your local IBM representative.

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide

To order DB2 publications from DB2 Marketing and Sales in the United States or Canada, call 1-800-IBM-4YOU (426-4968).

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

Contents

Part 1. Developing PHP applications 1

Chapter 1. Introduction to PHP application development for DB2 3

Chapter 2. Setting up the PHP environment 5
Setting up the PHP environment on Windows . . . 5
Setting up the PHP environment on Linux or UNIX . 5

Chapter 3. Developing with ibm_db2 . . 9
Connecting to a DB2 database with PHP (ibm_db2) . 9
Retrieving database metadata (ibm_db2) 9
Executing XQuery expressions in PHP (ibm_db2) . . 11
Executing SQL statements 12
Executing a single SQL statement in PHP (ibm_db2) 12
Preparing and executing SQL statements in PHP (ibm_db2) 13
Inserting large objects in PHP (ibm_db2) . . . 14
Reading query result sets 15
Fetching columns from result sets in PHP (ibm_db2) 15
Fetching rows from result sets in PHP (ibm_db2) 15
Fetching large objects in PHP (ibm_db2) 16
Managing transactions in PHP (ibm_db2) 17
Handling errors and warning messages (ibm_db2) 18
Calling stored procedures 19
Calling stored procedures with OUT or INOUT parameters in PHP (ibm_db2) 19
Calling stored procedures that return multiple result sets in PHP (ibm_db2) 20

Chapter 4. Developing with PDO. . . . 21
Connecting to a DB2 database with PHP (PDO) . . 21
Executing SQL statements 22
Executing a single SQL statement in PHP that returns no result sets (PDO) 22
Executing a single SQL statement in PHP that returns a result set (PDO). 22
Preparing and executing SQL statements (PDO) 23
Inserting large objects in PHP (PDO) 24
Reading query result sets 25
Fetching columns from result sets in PHP (PDO) 25
Fetching rows from result sets in PHP (PDO) . . 25
Fetching large objects in PHP (PDO) 27
Managing transactions in PHP (PDO). 28
Handling errors and warnings in PHP (PDO) . . . 28
Calling stored procedures 29
Calling stored procedures with OUT or INOUT parameters in PHP (PDO) 29
Calling stored procedures that return multiple result sets in PHP (PDO) 30

Part 2. Developing Perl Applications 31

Chapter 5. Programming Considerations for Perl 33
Database Connections in Perl 33
Fetching Results in Perl 33
Parameter Markers in Perl 34
SQLSTATE and SQLCODE Variables in Perl . . . 34
Perl Restrictions 35
Example of a Perl Program 35
Building Perl applications 35

Part 3. Appendixes. 37

Appendix A. Overview of the DB2 technical information 39
DB2 technical library in hardcopy or PDF format . . 39
Ordering printed DB2 books. 42
Displaying SQL state help from the command line processor 42
Accessing different versions of the DB2 Information Center 43
Displaying topics in your preferred language in the DB2 Information Center 43
Updating the DB2 Information Center installed on your computer or intranet server 44
DB2 tutorials 45
DB2 troubleshooting information 46
Terms and Conditions 46

Appendix B. Notices 49

Index 53

Part 1. Developing PHP applications

Chapter 1. Introduction to PHP application development for DB2

PHP: Hypertext Preprocessor (PHP) is an interpreted programming language primarily intended for the development of Web applications. The first version of PHP was created by Rasmus Lerdorf and contributed under an open source license in 1995. PHP was initially a very simple HTML templating engine, but over time the developers of PHP added database access functionality, rewrote the interpreter, introduced object-oriented support, and improved performance. Today, PHP has become a popular language for Web application development because of its focus on practical solutions and support for the most commonly required functionality in Web applications.

For the easiest install and configuration experience on Linux®, UNIX®, or Windows® operating systems, you can download and install Zend Core for IBM for use in production systems. Paid support for Zend Core for IBM is available from Zend. On Windows, precompiled binary versions of PHP are available for download from http://php.net/. Most Linux distributions include a precompiled version of PHP. On UNIX operating systems that do not include a precompiled version of PHP, you can compile your own version of PHP.

PHP is a modular language that enables you to customize the available functionality through the use of extensions. These extensions can simplify tasks such as reading, writing, and manipulating XML, creating SOAP clients and servers, and encrypting communications between server and browser. The most popular extensions for PHP, however, provide read and write access to databases so that you can easily create a dynamic database-driven Web site.

We have built on our existing PHP support by developing a new extension called pdo_ibm for anyone who wishes to use the PHP Application Objects (PDO) interface. This new extension along with the existing ibm_db2 extension will now be conveniently included as part of the IBM Data Server Client. The most up to date versions of ibm_db2 and pdo_ibm are available from the PHP Extension Community Library (PECL) http://pecl.php.net/. You can use either extension to access data stored in a DB2® database through your PHP application. The differences between the extensions are detailed as follows:

- ibm_db2 is an extension written, maintained, and supported by IBM® for access to DB2 databases. The ibm_db2 extension offers a procedural application programming interface (API) that, in addition to the normal create, read, update, and write database operations, also offers extensive access to the database metadata. You can compile the ibm_db2 extension with either PHP 4 or PHP 5.
- pdo_ibm is a driver for the PHP Data Objects (PDO) extension that offers access to DB2 databases through the standard object-oriented database interface introduced in PHP 5.1.

A third extension, Unified ODBC, has historically offered access to DB2 database systems. It is not recommended that you write new applications with this extension because ibm_db2 and pdo_ibm both offer significant performance and stability benefits over Unified ODBC. The ibm_db2 extension API makes porting an application that was previously written for Unified ODBC almost as easy as globally changing the `odbc_` function name to `db2_` throughout the source code of your application.

Chapter 2. Setting up the PHP environment

Setting up the PHP environment on Windows

DB2 supports database access for client applications written in the PHP programming language using either or both of the ibm_db2 extension and the pdo_ibm driver for the PHP Data Objects (PDO) extension. To install a binary version of PHP with support for DB2 on Windows, you can download and install the freely available Zend Core for IBM from http://zend.com/core/ibm/. However, you can also manually install the precompiled binary version of PHP on Windows.

The Apache HTTP Server must be installed on your system.

To install a precompiled version of PHP from http://www.php.net and enable support for on Windows:

1. Download the latest version of the PHP zip package and the collection of PECL modules zip package from http://www.php.net. The latest version of PHP at the time of writing is PHP 5.2.3.
2. Extract the PHP zip package into an install directory.
3. Extract the collection of PECL modules zip package into the `\ext\` subdirectory of your PHP installation directory.
4. Create a new file named `php.ini` in your installation directory by making a copy of the `php.ini-recommended` file.
5. Open the `php.ini` file in a text editor and add the following lines.
   - To enable the PDO extension and pdo_ibm driver:

     ```
     extension=php_pdo.dll
     extension=php_pdo_ibm.dll
     ```

   - To enable the ibm_db2 extension:

     ```
     extension=php_ibm_db2.dll
     ```

6. Enable PHP support in Apache HTTP Server 2.x by adding the following lines to your `httpd.conf` file, in which *phpdir* refers to the PHP install directory:

   ```
   LoadModule php5_module 'phpdir/php5apache2.dll'
   AddType application/x-httpd-php .php
   PHPIniDir 'phpdir'
   ```

7. Restart the Apache HTTP Server to enable the changed configuration.

Setting up the PHP environment on Linux or UNIX

DB2 supports database access for client applications written in the PHP programming language using either or both of the ibm_db2 extension and the pdo_ibm driver for the PHP Data Objects (PDO) extension. To install a binary version of PHP with support for DB2 on Linux or AIX, you can download and install the freely available Zend Core for IBM from http://zend.com/core/ibm/. However, you can also manually compile and install PHP from source.

- The Apache HTTP Server must be installed on your system.
- The DB2 development header files and libraries must be installed on your system.

- The gcc compiler and other development packages including apache-devel, autoconf, automake, bison, flex, gcc, and libxml2-devel package must be installed on your system.

To compile PHP from source with support for on Linux or UNIX:

1. Download the latest version of the PHP tarball from http://www.php.net. The latest version of PHP at the time of writing is PHP 5.2.3.
2. Untar the file by issuing the following command:

   ```
   tar -xjf php-5.x.x.tar.bz2
   ```
3. Change directories into the newly created `php-5.x.x` directory.
4. Configure the makefile by issuing the `configure` command. Specify the features and extensions you want to include in your custom version of PHP. A typical configure command includes the following options:

   ```
   ./configure --enable-cli --disable-cgi --with-apxs2=/usr/sbin/apxs2
   --with-zlib --with-pdo-ibm=<sqllib>
   ```

   The configure options have the following effects:

   --enable-cli
   Enables the command line mode of PHP access.

   --disable-cgi
   Disables the Common Gateway Interface (CGI) mode of PHP access.

   --with-apxs2=/usr/sbin/apxs2
   Enables the Apache 2 dynamic server object (DSO) mode of PHP access.

   --with-zlib
   Enables zlib compression support.

   --with-pdo-ibm=<sqllib>
   Enables the pdo_ibm driver using the DB2 CLI library to access database systems. The *<sqllib>* setting refers to the directory in which DB2 is installed.
5. Compile the files by issuing the `make` command.
6. Install the files by issuing the `make install` command. Depending on how you configured the PHP install directory using the `configure` command, you might need root authority to successfully issue this command. This should install the executable files and update the Apache HTTP Server configuration to support PHP.
7. Install the ibm_db2 extension by issuing the following command as a user with root authority:

   ```
   pecl install ibm_db2
   ```

   This command downloads, configure, compiles, and installs the ibm_db2 extension for PHP.
8. Copy the `php.ini-recommended` file to the configuration file path for your new PHP installation. To determine the configuration file path, issue the `php -i` command and look for the `php.ini` keyword. Rename the file to `php.ini`.
9. Open the new `php.ini` file in a text editor and add the following lines, where *instance* refers to the name of the DB2 instance on Linux or UNIX..
   - To set the DB2 environment for pdo_ibm:

     ```
     PDO_IBM.db2_instance_name=instance
     ```
   - (Linux or UNIX) To enable the ibm_db2 extension and set the DB2 environment:

```
extension=ibm_db2.so
ibm_db2.instance_name=instance
```

10. Restart the Apache HTTP Server to enable the changed configuration.

Chapter 3. Developing with ibm_db2

Connecting to a DB2 database with PHP (ibm_db2)

You must connect to a DB2 database before you can create, update, delete, or retrieve data from that data source. The ibm_db2 extension for PHP enables you to connect to a DB2 database using either a cataloged connection or a direct TCP/IP connection to the DB2 database management system. You can also create persistent connections to a database. Persistent connections improve performance by keeping the connection open between PHP requests and by reusing the connection when a subsequent PHP script requests a connection with an identical set of credentials.

Before connecting to a DB2 database through the ibm_db2 extension, you must set up the PHP environment on your system and enable the ibm_db2 extension.

1. Create a connection to a DB2 database:
   - To create a non-persistent connection to a DB2 database, call `db2_connect()` with a *database* value that specifies either a cataloged database name or a complete database connection string for a direct TCP/IP connection.
   - To create a persistent connection to a DB2 database, call `db2_pconnect()` with a *database* value that specifies either a cataloged database name or a complete database connection string for a direct TCP/IP connection.
2. Check the value returned by `db2_connect()` or `db2_pconnect`.
   - If the value returned by `db2_connect()` or `db2_pconnect` is FALSE, the connection attempt failed. You can retrieve diagnostic information through `db2_conn_error()` and `db2_conn_errormsg()`.
   - If the value returned by `db2_connect()` or `db2_pconnect` is not FALSE, the connection attempt succeeded. You can use the connection resource to create, update, delete, or retrieve data with other ibm_db2 functions.

When you create a connection by calling `db2_connect()`, PHP closes the connection to the database:

- When you call `db2_close()` for the connection,
- When you set the connection resource to NULL,
- Or when the PHP script finishes.

When you create a connection by calling `db2_pconnect()`, PHP ignores any calls to `db2_close()` for the specified connection resource and keeps the connection to the database open for subsequent PHP scripts.

Retrieving database metadata (ibm_db2)

Some classes of applications, such as administration interfaces, need to dynamically reflect the structure and SQL objects contained in arbitrary databases. One approach to retrieving metadata about a database is to issue SELECT statements directly against the system catalog tables; however, the schema of the system catalog tables may change between versions of DB2, or the schema of the system catalog tables on DB2 Database for Linux, UNIX, and Windows may differ from the schema of the system catalog tables on DB2 for z/OS®. Rather than laboriously maintaining these differences in your application code, the ibm_db2 extension for PHP offers a standard set of functions that return metadata for

databases served by DB2 Database for Linux, UNIX, and Windows, Cloudscape™, and, through DB2 Connect™, DB2 for z/OS and DB2 for i5/OS®.

- You must set up the PHP environment on your system and enable the ibm_db2 extension.
- You must have a connection resource returned from `db2_connect()` or `db2_pconnect()`.

1. Call the function that returns the metadata which you require:

   `db2_client_info()`
   Returns metadata about the DB2 client software and configuration.

   `db2_column_privileges()`
   Lists the columns and associated privileges for a table.

   `db2_columns()`
   Lists the columns and associated metadata for a table.

   `db2_foreign_keys()`
   Lists the foreign keys for a table.

   `db2_primary_keys()`
   Lists the primary keys for a table.

   `db2_procedure_columns()`
   Lists the parameters for one or more stored procedures.

   `db2_procedures()`
   Lists the stored procedures registered in the database.

   `db2_server_info()`
   Returns metadata about the database management system software and configuration.

   `db2_special_columns()`
   Lists the unique row identifiers for a table.

   `db2_statistics()`
   Lists the indexes and statistics for a table.

   `db2_table_privileges()`
   Lists tables and their associated privileges in the database.

   Note that while most of the ibm_db2 metadata functions accept a qualifier or catalog parameter, this parameter should only be set to a non-NULL value when you are connected to .

2. Depending on which metadata function you called,
   - The `db2_client_info()` and `db2_server_info()` functions directly return a single object with read-only properties. You can use the properties of these objects to create an application that behaves differently depending on the database management system to which it connects. For example, rather than encoding a limit of the lowest common denominator for all possible database management systems, a Web-based database administration application built on the ibm_db2 extension could use the `db2_server_info()->MAX_COL_NAME_LEN` property to dynamically display text fields for naming columns with maximum lengths that correspond to the maximum length of column names on the database management system to which it is connected.
   - The other metadata functions return result sets with columns defined for each function. Retrieve rows from the result set using the normal ibm_db2 functions for this purpose.

Note that calling metadata functions consumes a significant amount of database management system resources. If possible, consider caching the results of your calls for subsequent usage.

Executing XQuery expressions in PHP (ibm_db2)

After connecting to a DB2 database, your PHP script is ready to issue XQuery expressions. The `db2_exec()` and `db2_execute()` functions execute SQL statements, through which you can pass your XQuery expressions. A typical use of `db2_exec()` is to set the default schema for your application in a common include file or base class.

You must set up the PHP environment on your system and enable the ibm_db2 extension.

To avoid the security threat of injection attacks, `db2_exec()` should only be used to execute SQL statements composed of static strings. Interpolation of PHP variables representing user input into the XQuery expression can expose your application to injection attacks.

1. Call `db2_exec()` with the following arguments:
   a. The connection resource;
   b. A string containing the SQL statement, including the XQuery expression. The XQuery expression needs to be wrapped in a XMLQUERY clause in the SQL statement.
   c. (Optional): an array containing statement options

      DB2_ATTR_CASE
      For compatibility with database systems that do not follow the SQL standard, this option sets the case in which column names will be returned to the application. By default, the case is set to *DB2_CASE_NATURAL*, which returns column names as they are returned by DB2. You can set this parameter to *DB2_CASE_LOWER* to force column names to lower case, or to *DB2_CASE_UPPER* to force column names to upper case.

      DB2_ATTR_CURSOR
      This option sets the type of cursor that ibm_db2 returns for result sets. By default, ibm_db2 returns a forward-only cursor (*DB2_FORWARD_ONLY*) which returns the next row in a result set for every call to `db2_fetch_array()`, `db2_fetch_assoc()`, `db2_fetch_both()`, `db2_fetch_object()`, or `db2_fetch_row()`. You can set this parameter to *DB2_SCROLLABLE* to request a scrollable cursor so that the ibm_db2 fetch functions accept a second argument specifying the absolute position of the row that you want to access within the result set .
2. Check the value returned by `db2_exec()`:
   - If the value is FALSE, the SQL statement failed. You can retrieve diagnostic information through the `db2_stmt_error()` and `db2_stmt_errormsg()` functions.
   - If the value is not FALSE, the SQL statement succeeded and returned a statement resource that can be used in subsequent function calls related to this query.

```
<?php
$xquery = '$doc/customerinfo/phone';
$stmt = db2_exec($conn, "select xmlquery('$xquery'
PASSING INFO AS \"doc\") from customer");?>
```

Executing SQL statements

Executing a single SQL statement in PHP (ibm_db2)

After connecting to a DB2 database, most PHP scripts will execute one or more SQL statements. The `db2_exec()` function executes a single SQL statement that accepts no input parameters. A typical use of `db2_exec()` is to set the default schema for your application in a common include file or base class.

You must set up the PHP environment on your system and enable the ibm_db2 extension.

To avoid the security threat of SQL injection attacks, `db2_exec()` should only be used to execute SQL statements composed of static strings. Interpolation of PHP variables representing user input into the SQL statement can expose your application to SQL injection attacks.

1. Call `db2_exec()` with the following arguments:
   a. The connection resource;
   b. A string containing the SQL statement;
   c. (Optional): an array containing statement options

      DB2_ATTR_CASE
      For compatibility with database systems that do not follow the SQL standard, this option sets the case in which column names will be returned to the application. By default, the case is set to *DB2_CASE_NATURAL*, which returns column names as they are returned by DB2. You can set this parameter to *DB2_CASE_LOWER* to force column names to lower case, or to *DB2_CASE_UPPER* to force column names to upper case.

      DB2_ATTR_CURSOR
      This option sets the type of cursor that ibm_db2 returns for result sets. By default, ibm_db2 returns a forward-only cursor (*DB2_FORWARD_ONLY*) which returns the next row in a result set for every call to `db2_fetch_array()`, `db2_fetch_assoc()`, `db2_fetch_both()`, `db2_fetch_object()`, or `db2_fetch_row()`. You can set this parameter to *DB2_SCROLLABLE* to request a scrollable cursor so that the ibm_db2 fetch functions accept a second argument specifying the absolute position of the row that you want to access within the result set .
2. Check the value returned by `db2_exec()`:
   - If the value is FALSE, the SQL statement failed. You can retrieve diagnostic information through the `db2_stmt_error()` and `db2_stmt_errormsg()` functions.
   - If the value is not FALSE, the SQL statement succeeded and returned a statement resource that can be used in subsequent function calls related to this query.

If the SQL statement selected rows using a scrollable cursor, or inserted, updated, or deleted rows, you can call `db2_num_rows()` to return the number of rows that the statement returned or affected. If the SQL statement returned a result set, you can begin fetching rows.

Preparing and executing SQL statements in PHP (ibm_db2)

Most SQL statements in PHP applications use variable input to determine the results of the SQL statement. To pass user-supplied input to an SQL statement safely, prepare a statement using parameter markers (?) representing the variable input. When you execute the prepared statement, you bind input values to the parameter markers. The database engine ensures that each input value is treated as a single parameter, preventing SQL injection attacks against your application. Compared to statements issued through `db2_exec()`, prepared statements offer a performance advantage because the database management system creates an access plan for each prepared statement that it can reuse if the statement is reissued subsequently.

You must set up the PHP environment on your system and enable the ibm_db2 extension.

You can only use parameter markers as a place holder for column or predicate values. The SQL compiler would be unable to create an access plan for a statement that used parameter markers in place of column names, table names, or other SQL identifiers.

To prepare and execute an SQL statement:

1. Call `db2_prepare()` with the following arguments:
   a. The connection resource
   b. A string containing the SQL statement, including parameter markers (?) for any column or predicate values that require variable input
   c. (Optional): An array containing statement options

      DB2_ATTR_CASE
      For compatibility with database systems that do not follow the SQL standard, this option sets the case in which column names will be returned to the application. By default, the case is set to *DB2_CASE_NATURAL*, which returns column names as they are returned by DB2. You can set this parameter to *DB2_CASE_LOWER* to force column names to lower case, or to *DB2_CASE_UPPER* to force column names to upper case.

      DB2_ATTR_CURSOR
      This option sets the type of cursor that ibm_db2 returns for result sets. By default, ibm_db2 returns a forward-only cursor (*DB2_FORWARD_ONLY*) which returns the next row in a result set for every call to `db2_fetch_array()`, `db2_fetch_assoc()`, `db2_fetch_both()`, `db2_fetch_object()`, or `db2_fetch_row()`. You can set this parameter to *DB2_SCROLLABLE* to request a scrollable cursor so that the ibm_db2 fetch functions accept a second argument specifying the absolute position of the row that you want to access within the result set.
2. Check the value returned by `db2_prepare()`.

- If the value is FALSE, the SQL statement failed. You can retrieve diagnostic information through the db2_stmt_error() and db2_stmt_errormsg() functions.
- If the value is not FALSE, the SQL statement succeeded and returned a statement resource that can be used in subsequent function calls related to this query.

3. (Optional): Call db2_bind_param() for each parameter marker in the SQL statement with the following arguments:
   a. The statement resource
   b. An integer representing the position of the parameter marker in the SQL statement
   c. The value to use in place of the parameter marker
4. Call db2_execute with the following arguments:
   a. The statement resource
   b. (Optional): An array containing the values to use in place of the parameter markers, in order

```
$sql = "SELECT firstnme, lastname FROM employee WHERE bonus > ? AND bonus < ?";
$stmt = db2_prepare($conn, $sql);
if (!$stmt) {
  // Handle errors
}

// Explicitly bind parameters
db2_bind_param($stmt, 1, $_POST['lower']);
db2_bind_param($stmt, 2, $_POST['upper']);

db2_execute($stmt);
// Process results

// Invoke prepared statement again using dynamically bound parameters
db2_execute($stmt, array($_POST['lower'], $_POST['upper']);
```

If you execute a prepared statement that returns one or more result sets, you can begin retrieving rows from the statement resource by calling the db2_fetch_array() , db2_fetch_assoc(), db2_fetch_both(), db2_fetch_object(), or db2_fetch_row() functions.

Inserting large objects in PHP (ibm_db2)

The ibm_db2 extension supports the entire range of DB2 data types, including character large object (CLOB) and binary large object (BLOB) data types. When you insert a large object into a database, you can treat the large object simply as a PHP string. However, treating a large object as a PHP string is an approach that consumes more resources on your PHP server than necessary. Rather than loading all of the data for a large object into a PHP string, and then passing that to DB2 through an INSERT statement, you can insert large objects directly from a file on your PHP server.

You must set up the PHP environment on your system and enable the ibm_db2 extension.

To insert a large object into the database directly from a file:

1. Call db2_prepare() to prepare an INSERT statement with a parameter marker representing the large object column.

2. Set the value of a PHP variable to the path and name of the file that contains the data for the large object. The path can be relative or absolute, and is subject to the access permissions of the PHP executable.
3. Call db2_bind_param() to bind the parameter marker to the file that contains the data for the large object. The third parameter is a string representing the name of the PHP variable that holds the name of the file containing the data for the large object. The fourth parameter is DB2_PARAM_FILE, which tells the ibm_db2 extension to retrieve the data from a file.
4. Call db2_execute() to issue the INSERT statement and bind the data from the file into the database.

```
$stmt = db2_prepare($conn, "INSERT INTO animal_pictures(picture) VALUES (?)");

$picture = "/opt/albums/spook/grooming.jpg";
$rc = db2_bind_param($stmt, 1, "picture", DB2_PARAM_FILE);
$rc = db2_execute($stmt);
```

Reading query result sets

Fetching columns from result sets in PHP (ibm_db2)

When you execute a statement that returns one or more result sets, you usually need to iterate through the returned rows of each result set. If your result set includes columns with extremely large data (such as a column defined with a BLOB or CLOB data type), you might prefer to retrieve the data on a column-by-column basis to avoid using too much memory in your PHP process.

- You must set up the PHP environment on your system and enable the ibm_db2 extension.
- You must have a statement resource returned from db2_exec() or db2_execute() with one or more associated result sets.

1. Call the db2_fetch_row() function to advance the cursor to the next row in the result set. The first time you call a fetch function for a given result set advances the cursor to the first row of the result set. If you requested a scrollable cursor, you can also specify the number of the row in the result set that you want to retrieve.
2. Check the result returned by db2_fetch_row(). If the result is FALSE, there are no more rows in the result set.
3. Call the db2_result() function to retrieve the value from the requested column by passing either an integer representing the position of the column in the row (starting with 0 for the first column), or a string representing the name of the column.

```
<?php
$sql = 'SELECT name, breed FROM animals WHERE weight < ?';
$stmt = db2_prepare($conn, $sql);
db2_execute($stmt, array(10));
while (db2_fetch_row($stmt)) {
    $name = db2_result($stmt, 0);
    $breed = db2_result($stmt, 'BREED');
    print "$name $breed";
}
?>
```

Fetching rows from result sets in PHP (ibm_db2)

When you execute a statement that returns one or more result sets, you usually need to iterate through the returned rows.

- You must set up the PHP environment on your system and enable the ibm_db2 extension.
- You must have a statement resource returned from `db2_exec()` or `db2_execute()` with one or more associated result sets.

Call the ibm_db2 fetch function that returns the data from the row in the format you prefer:

db2_fetch_array()
: Returns an array containing the data corresponding to the columns of the row indexed by column position starting at 0

db2_fetch_assoc()
: Returns an array containing the data corresponding to the columns of the row indexed by column name.

db2_fetch_both()
: Returns an array containing the data corresponding to the columns of the row indexed by both column name and by column position starting at 0.

db2_fetch_object()
: Returns an object containing the data from the row. The object holds properties matching the column names of the row which, when accessed, return the corresponding values of the columns.

You must pass the statement resource as the first argument. If you requested a scrollable cursor when you executed `db2_exec()` or `db2_prepare()`, you can pass an absolute row number as the second argument. With the default forward-only cursor, each call to a fetch method returns the next row in the result set. You can continue fetching rows until the fetch method returns `FALSE`, which signifies that you have reached the end of the result set.

```
$stmt = db2_exec($conn, "SELECT firstnme, lastname FROM employee");
while ($row = db2_fetch_object($stmt)) {
  print "Name: <p>{$row->FIRSTNME} {$row->LASTNAME}</p>";
}
```

Fetching large objects in PHP (ibm_db2)

The ibm_db2 extension supports the entire range of DB2 data types, including character large object (CLOB) and binary large object (BLOB) data types. When you fetch a large object from a result set, you can treat the large object simply as a PHP string. However, treating a large object as a PHP string is an approach that consumes more resources on your PHP server than necessary. If your ultimate goal is to create a file that contains the data for a large object, you can save system resources by fetching large objects directly into a file on your PHP server.

You must set up the PHP environment on your system and enable the ibm_db2 extension.

To fetch a large object from the database directly into a file:

1. Create a PHP variable representing a stream. For example, the return value from a call to `fopen()`.
2. Call `db2_prepare()` to create a SELECT statement.
3. Call `db2_bind_param()` to bind the output column for the large object to the PHP variable representing the stream. The third parameter is a string representing the name of the PHP variable that holds the name of the file that

is to contain the data from the large object. The fourth parameter is DB2_PARAM_FILE, which tells the ibm_db2 extension to write the data into a file.

4. Call db2_execute() to issue the SQL statement.
5. Call an ibm_db2 fetch function of your choice (for example, db2_fetch_object()), to retrieve the next row in the result set.

```
$stmt = db2_prepare($conn, "SELECT name, picture FROM animal_pictures");
$picture = fopen("/opt/albums/spook/grooming.jpg", "wb");
$rc = db2_bind_param($stmt, 1, "nickname", DB2_CHAR, 32);
$rc = db2_bind_param($stmt, 2, "picture", DB2_PARAM_FILE);
$rc = db2_execute($stmt);
$rc = db2_fetch_object($stmt);
```

Managing transactions in PHP (ibm_db2)

By default, the ibm_db2 extension opens every connection in autocommit mode. Autocommit mode helps prevent locking escalation issues that can impede the performance of highly scalable Web applications. In some scripts, however, you might need to roll back a transaction containing one or more SQL statements. The ibm_db2 extension enables you to exert fine-grained control over your transactions.

You must set up the PHP environment on your system and enable the ibm_db2 extension.

You must use a regular connection created with db2_connect() to control database transactions in PHP. Persistent connections always use autocommit mode.

To begin a transaction:

1. Create a database connection using the "AUTOCOMMIT" => DB2_AUTOCOMMIT_OFF setting in the db2_connect() options array. You can also turn autocommit off for an existing connection resource by calling db2_autocommit($conn, DB2_AUTOCOMMIT_OFF). Calling db2_autocommit() requires additional communication from PHP to the database management system and may affect the performance of your PHP scripts.
2. Issue one or more SQL statements within the scope of the database transaction using the connection resource for which transactions have been enabled.
3. Commit or rollback the transaction:
   - To commit the transaction, call db2_commit().
   - To rollback the transaction, call db2_rollback().
4. (Optional): Return the database connection to autocommit mode by calling db2_autocommit($conn, DB2_AUTOCOMMIT_ON). If you issue another SQL statement without returning the database connection to autocommit mode, you begin a new transaction that will require a commit or rollback.

If you issue SQL statements in a transaction and the script ends without explicitly committing or rolling back the transaction, the ibm_db2 extension automatically rolls back any work performed in the transaction.

```
$conn = db2_connect('SAMPLE', 'db2inst1', 'ibmdb2', array(
  'AUTOCOMMIT' => DB2_AUTOCOMMIT_ON));

// Issue one or more SQL statements within the transaction
$result = db2_exec($conn, 'DELETE FROM TABLE employee');
if ($result === FALSE) {
  print '<p>Unable to complete transaction!</p>';
  db2_rollback($conn);
}
```

```
else {
  print '<p>Successfully completed transaction!</p>';
  db2_commit($conn);
}
```

Handling errors and warning messages (ibm_db2)

Problems occasionally happen when you attempt to connect to a database or issue an SQL statement. The password for your connection might be incorrect, the table you referred to in a SELECT statement might not exist, or the syntax for an SQL statement might be invalid. You need to code defensively and use the error-handling functions offered by the ibm_db2 extension to enable your application to recover gracefully from a problem.

You must set up the PHP environment on your system and enable the ibm_db2 extension.

1. Check the value returned from the ibm_db2 function to ensure the function returned successfully. If the function can return the value 0, such as `db2_num_rows()`, you must explicitly test whether the value was `FALSE` using PHP's `===` operator.
2. If the function returned `FALSE` instead of the connection resource, statement resource, or numeric value you expected, call the ibm_db2 error handling function appropriate to the application context and the needs of your application:

   Connection errors

   To retrieve the SQLSTATE returned by the last connection attempt, call `db2_conn_error()`. To retrieve a descriptive error message appropriate for an application error log, call `db2_conn_errormsg()`.

   ```
   $connection = db2_connect($database, $user, $password);
   if (!$connection) {
       $this->state = db2_conn_error();
       return false;
   }
   ```

   SQL errors (executing SQL statements directly and fetching results)

   To retrieve the SQLSTATE returned by the last attempt to prepare or execute an SQL statement, or to fetch a result from a result set, call `db2_stmt_error()`. To retrieve a descriptive error message appropriate for an application error log, call `db2_stmt_errormsg()`.

   ```
   $stmt = db2_prepare($connection, "DELETE FROM employee
   WHERE firstnme = ?");
   if (!$stmt) {
       $this->state = db2_stmt_error();
       return false;
   }
   ```

   SQL errors (executing prepared statements)

   If `db2_prepare()` returned successfully, but a subsequent call to `db2_execute()` fails, call `db2_stmt_error()` or `db2_stmt_errormsg()` and pass the resource returned from the call to `db2_prepare()` as the argument.

   ```
   $success = db2_execute($stmt, array('Dan');
   if (!$success) {
       $this->state = db2_stmt_error($stmt);
       return $false;
   }
   ```

3. To avoid the possibility of security vulnerabilities resulting from directly displaying the raw SQLSTATE returned from the database, and to offer a better overall user experience in your Web application, use a `switch` structure to recover from known error states or return custom error messages.

```
switch($this->state):
    case '22001':
        // More data than allowed for the defined column
        $message = "You entered too many characters for this value.";
        break;
```

Calling stored procedures

Calling stored procedures with OUT or INOUT parameters in PHP (ibm_db2)

DB2 supports stored procedures with parameters that only accept an input value (IN parameters), that only return an output value (OUT parameters), or that accept an input value and return an output value (INOUT). With the ibm_db2 extension for PHP you can handle IN parameters like any other parameter marker in an SQL statement. However, the ibm_db2 extension also enables you to CALL stored procedures with OUT and INOUT parameters and retrieve the output values from those parameters.

You must set up the PHP environment on your system and enable the ibm_db2 extension.

To call a stored procedure with OUT or INOUT parameters:

1. Call `db2_prepare()` to prepare a CALL statement with parameter markers representing the OUT and INOUT parameters.
2. Call `db2_bind_param()` to bind each parameter marker to the name of the PHP variable that will hold the output value of the parameter after the CALL statement has been issued. For INOUT parameters, the value of the PHP variable is passed as the input value of the parameter when the CALL statement is issued. Set the fourth parameter for `db2_bind_param()` to either `DB2_PARAM_OUT`, representing an OUT parameter, or `DB2_PARAM_INOUT`, representing an INOUT parameter.
3. Call `db2_execute()` to issue the CALL statement and bind the data from the stored procedure into the PHP variables.

```
$sql = 'CALL match_animal(?, ?)';
$stmt = db2_prepare($conn, $sql);

$second_name = "Rickety Ride";
$weight = 0;

db2_bind_param($stmt, 1, "second_name", DB2_PARAM_INOUT);
db2_bind_param($stmt, 2, "weight", DB2_PARAM_OUT);

print "Values of bound parameters _before_ CALL:\n";
print "  1: {$second_name} 2: {$weight}\n";

db2_execute($stmt);

print "Values of bound parameters _after_ CALL:\n";
print "  1: {$second_name} 2: {$weight}\n";
```

Calling stored procedures that return multiple result sets in PHP (ibm_db2)

DB2 enables you to create and call stored procedures that return more than one result set. The ibm_db2 extension for PHP fully supports this capability through the `db2_next_result()` function. You can use this function to fetch rows from different result sets returned by a single call to the same stored procedure in any order you prefer.

- You must set up the PHP environment on your system and enable the ibm_db2 extension.
- You must have a statement resource returned from calling a stored procedure with `db2_exec()` or `db2_execute()`.

To return multiple result sets from a stored procedure:

1. The first result set is associated with the statement resource returned by the CALL statement.
2. Pass the original statement resource as the first argument to `db2_next_result()` to retrieve the second and subsequent result sets. This function returns `FALSE` when no more result sets are available.

```
$stmt = db2_exec($conn, 'CALL multiResults()');

print "Fetching first result set\n";
while ($row = db2_fetch_array($stmt)) {
    // work with row
}

print "\nFetching second result set\n";
$result_2 = db2_next_result($stmt);
if ($result_2) {
  while ($row = db2_fetch_array($result_2)) {
    // work with row
  }
}

print "\nFetching third result set\n";
$result_3 = db2_next_result($stmt);
if ($result_3) {
  while ($row = db2_fetch_array($result_3)) {
    // work with row
  }
}
```

Chapter 4. Developing with PDO

Connecting to a DB2 database with PHP (PDO)

You must connect to a DB2 database before you can create, update, delete, or retrieve data from that data source. The PHP Data Objects (PDO) interface for PHP enables you to connect to a DB2 database using either a cataloged connection or a direct TCP/IP connection to the DB2 database management system through the PDO_IBM extension. You can also create persistent connections to a data source that improve performance by keeping the connection open between PHP requests and reusing the connection when a subsequent PHP script requests a connection with an identical set of credentials.

You must set up the PHP 5.1 or higher environment on your system and enable the PDO and PDO_IBM extensions.

1. Create a connection to the DB2 database by calling the PDO constructor within a `try{}` block. Pass a *DSN* value that specifies `ibm:` for the PDO_IBM extension, followed by either a cataloged database name or a complete database connection string for a direct TCP/IP connection.
   - (Windows): By default, PDO_IBM uses connection pooling to minimize connection resources and improve connection performance.
   - (Linux and UNIX): PDO_IBM offers persistent connections if you pass `array(PDO::ATTR_PERSISTENT => TRUE)` as the fourth argument to the PDO constructor.
2. (Optional): Set error handling options for the PDO connection in the fourth argument to the PDO constructor:
   - by default, PDO sets an error message that can be retrieved through `PDO::errorInfo()` and an SQLCODE that can be retrieved through `PDO::errorCode()` when any error occurs; to request this mode explicitly, set `PDO::ATTR_ERRMODE => PDO::ERRMODE_SILENT`
   - to issue a PHP `E_WARNING` when any error occurs, in addition to setting the error message and SQLCODE, set `PDO::ATTR_ERRMODE => PDO::ERRMODE_WARNING`
   - to throw a PHP exception when any error occurs, set `PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION`
3. Catch any exception thrown by the `try{}` block in a corresponding `catch {}` block.

```
try {
  $connection = new PDO("ibm:SAMPLE", "db2inst1", "ibmdb2", array(
    PDO::ATTR_PERSISTENT => TRUE,
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION)
  );
}
catch (Exception $e) {
  echo($e->getMessage());
}
```

When you create a connection through PDO, PHP closes the connection to the database:

- when you set the PDO object to NULL,
- or when the PHP script finishes.

Executing SQL statements

Executing a single SQL statement in PHP that returns no result sets (PDO)

After connecting to a DB2 database, most PHP scripts will execute one or more SQL statements. The `PDO::exec()` method executes a single SQL statement that accepts no input parameters and returns no result set. A typical use of `PDO::exec()` is to set the default schema for your application in a common include file or base class.

You must set up the PHP environment on your system and enable the PDO_IBM extension.

To avoid the security threat of SQL injection attacks, `PDO::exec()` should only be used to execute SQL statements composed of static strings. Interpolation of PHP variables representing user input into the SQL statement can expose your application to SQL injection attacks.

To execute a single SQL statement in PHP:

1. Call the `PDO::exec()` method on the PDO connection object with a string containing the SQL statement.
2. If the SQL statement inserted, modified, or deleted rows, `PDO::exec()` returns an integer value representing the number of rows that were inserted, modified, or deleted. To determine if `PDO::exec()` returned FALSE indicating an error condition or 0 indicating that no rows were inserted, modified, or deleted, you must use the === operator to strictly test the returned value against FALSE.

```
$conn = new PDO('ibm:SAMPLE', 'db2inst1', 'ibmdb2');
$result = $conn->exec('SET SCHEMA myapp');
if ($result === FALSE) {
  print "Failed to set schema: " . $conn->errorMsg();
}
```

Executing a single SQL statement in PHP that returns a result set (PDO)

After connecting to a DB2 database, most PHP scripts will execute one or more SQL statements. The `PDO::query()` method executes a single SQL statement that accepts no input parameters and returns one or more result sets. A typical use of `PDO::query()` is to execute a static SELECT statement.

You must set up the PHP environment on your system and enable the PDO_IBM extension.

To avoid the security threat of SQL injection attacks, `PDO::query()` should only be used to execute SQL statements composed of static strings. Interpolation of PHP variables representing user input into the SQL statement can expose your application to SQL injection attacks.

To execute a single SQL statement in PHP that returns a result set:

1. Call the `PDO::query()` method on the PDO connection object with a string containing the SQL statement.
2. Check the value returned by `PDO::query()`.

- If the value is FALSE, the SQL statement failed. You can retrieve diagnostic information through the PDO::errorCode() and PDO::errorInfo() methods.
- If the value is not FALSE, the SQL statement succeeded and returned a PDOStatement resource that can be used in subsequent method calls.

```
$conn = new PDO('ibm:SAMPLE', 'db2inst1', 'ibmdb2');
$result = $conn->query('SELECT firstnme, lastname FROM employee');
if (!$result) {
  print "<p>Could not retrieve employee list: " . $conn->errorMsg(). "</p>";
}
while ($row = $conn->fetch()) {
  print "<p>Name: {$row[0] $row[1]}</p>";
}
```

After creating a PDOStatement object with PDO::query(), you can immediately begin retrieving rows from the object with the PDOStatement::fetch() or PDOStatement::fetchAll() methods.

Preparing and executing SQL statements (PDO)

Most SQL statements in PHP applications use variable input to determine the results of the SQL statement. To pass user-supplied input to an SQL statement safely, prepare a statement using parameter markers (?) or named variables representing the variable input. When you execute the prepared statement, you bind input values to the parameter markers. The database engine ensures that each input value is treated as a single parameter, preventing SQL injection attacks against your application. Compared to statements issued through PDO::exec(), prepared statements offer a performance advantage because the database management system creates an access plan for each prepared statement that it can reuse if the statement is reissued subsequently.

You must set up the PHP environment on your system and enable the PDO_IBM extension.

- You can only use parameter markers as a place holder for column or predicate values. The SQL compiler would be unable to create an access plan for a statement that used parameter markers in place of column names, table names, or other SQL identifiers.
- You cannot use both question mark parameter markers (?) and named parameter markers (:name) in the same SQL statement.

To prepare and execute an SQL statement:

1. Call PDO::prepare() with the following arguments:
   a. A string containing the SQL statement including either parameter markers (?) or named variables (:name) for any column or predicate values that require variable input
   b. (Optional): An array containing statement options

      PDO::ATTR_CURSOR
      This option sets the type of cursor that PDO returns for result sets. By default, PDO returns a forward-only cursor (*PDO::CURSOR_FWDONLY*) which returns the next row in a result set for every call to PDOStatement::fetch(). You can set this parameter to *PDO::CURSOR_SCROLL* to request a scrollable cursor.
2. Check the value returned by PDO::prepare().
   - If the value is FALSE, the SQL statement failed. You can retrieve diagnostic information through the PDO::errorCode() and PDO::errorInfo() methods.

- If the value is not FALSE, the SQL statement succeeded and returned a PDOStatement object that can be used in subsequent method calls.

3. (Optional): Call `PDOStatement::bindParam()` for each parameter marker in the SQL statement with the following arguments:
   a. The parameter identifier. For question mark parameter markers (`?`), this is an integer representing the 1-indexed position of the parameter in the SQL statement. For named parameter markers (`:name`), this is a string representing the parameter name.
   b. The value to use in place of the parameter marker
4. Call `PDOStatement::execute()`, optionally passing an array containing the values to use in place of the parameter markers, either in order for question mark parameter markers, or as a `:name => value` associative array for named parameter markers.

```
$sql = "SELECT firstnme, lastname FROM employee WHERE bonus > ? AND bonus < ?";
$stmt = $conn->prepare($sql);
if (!$stmt) {
  // Handle errors
}

// Explicitly bind parameters
$stmt->bindParam(1, $_POST['lower']);
$stmt->bindParam(2, $_POST['upper']);

$stmt->execute($stmt);

// Invoke statement again using dynamically bound parameters
$stmt->execute($stmt, array($_POST['lower'], $_POST['upper']);
```

If you successfully execute a prepared statement that returns one or more result sets, you can begin retrieving rows from the statement resource by calling the `PDOStatement::fetch()` or `PDOStatement::fetchAll()` methods.

Inserting large objects in PHP (PDO)

The PDO extension supports the entire range of DB2 data types, including character large object (CLOB) and binary large object (BLOB) data types. When you insert a large object into a database, you can treat the large object simply as a PHP string. However, treating a large object as a PHP string is an approach that consumes more resources on your PHP server than necessary. Rather than loading all of the data for a large object into a PHP string, and then passing that to DB2 through an INSERT statement, you can insert large objects directly from a file on your PHP server.

You must set up the PHP 5.1 or higher environment on your system and enable the PDO and PDO_IBM extensions.

To insert a large object into the database directly from a file:

1. Call `PDO::prepare()` to create a PDOStatement object from an INSERT statement with a parameter marker representing the large object column.
2. Create a PHP variable representing a stream–for example, the return value from a call to `fopen()`.
3. Call `PDOStatement::bindParam()` to bind the parameter marker to the PHP variable representing the stream of data for the large object. The third parameter is a string representing the name of the PHP variable that holds the

name of the file containing the data for the large object. The fourth parameter is a PHP constant, PDO::PARAM_LOB, which tells the PDO extension to retrieve the data from a file.

4. Call PDOStatement::execute() to issue the INSERT statement and bind the data from the file into the database.

```
$stmt = $conn->prepare("INSERT INTO animal_pictures(picture) VALUES (?)");
$picture = fopen("/opt/albums/spook/grooming.jpg", "rb");
$stmt->bindParam($stmt, 1, $picture, PDO::PARAM_LOB);
$stmt->execute();
```

Reading query result sets

Fetching columns from result sets in PHP (PDO)

When you execute a statement that returns one or more result sets, you usually need to iterate through the returned rows of each result set. In some cases, you only need to return a single column from each row in the result set. While you could rewrite a SELECT statement for that purpose, you might not have the privileges required to rewrite a stored procedure that returns more columns than you require.

- You must set up the PHP environment on your system and enable the PDO and PDO_IBM extensions.
- You must have a statement resource returned from PDO::query() or PDOStatement::execute() with one or more associated result sets.

If you decide to fetch a column from a row, instead of retrieving all of the columns in the entire row simultaneously, you can only return a single column from each row.

- To return a single column from a single row in the result set:

  Call the PDOStatement::fetchColumn() method, specifying the column you want to retrieve as the first argument of the method. Column numbers start at 0. If you do not specify a column, PDOStatement::fetchColumn() returns the first column in the row.
- To return an array containing a single column from all of the remaining rows in the result set:

  Call the PDOStatement::fetchAll() method, passing PDO::FETCH_COLUMN as the first argument, and the column you want to retrieve as the second argument, to return an array of the values for the selected column from the result set. Column numbers start at 0. If you do not specify a column, PDOStatement::fetchAll(PDO::FETCH_COLUMN) returns the first column in the row.

Fetching rows from result sets in PHP (PDO)

When you execute a statement that returns one or more result sets, you usually need to iterate through the returned rows.

- You must set up the PHP environment on your system and enable the PDO extension.
- You must have a PDOStatement object returned from PDO::query() or PDOStatement::execute() with one or more associated result sets.

To return a single row from a result set as an array or object, call the PDOStatement::fetch() method.

To return all of the rows from the result set as an array of arrays or objects, call the `PDOStatement::fetchAll()` method.

By default, PDO returns each row as an array indexed by column name and 0-indexed column position in the row. You can request a different return style by passing one of the following constants as the first parameter of `PDOStatement::fetch()`:

PDO::FETCH_ASSOC
: Returns an array indexed by column name as returned in your result set.

PDO::FETCH_BOTH (default)
: Returns an array indexed by both column name and 0-indexed column number as returned in your result set

PDO::FETCH_BOUND
: Returns TRUE and assigns the values of the columns in your result set to the PHP variables to which they were bound with the `PDOStatement::bindParam()` method.

PDO::FETCH_CLASS
: Returns a new instance of the requested class, mapping the columns of the result set to named properties in the class.

PDO::FETCH_INTO
: Updates an existing instance of the requested class, mapping the columns of the result set to named properties in the class.

PDO::FETCH_LAZY
: Combines `PDO::FETCH_BOTH` and `PDO::FETCH_OBJ`, creating the object variable names as they are accessed.

PDO::FETCH_NUM
: Returns an array indexed by column number as returned in your result set, starting at column 0.

PDO::FETCH_OBJ
: Returns an anonymous object with property names that correspond to the column names returned in your result set.

(Optional): If you requested a scrollable cursor when you called `PDO::query()` or `PDOStatement::execute()`, you can pass two more arguments to `PDOStatement::fetch()`:

1. The fetch orientation for this fetch request:

   PDO::FETCH_ORI_NEXT (default)
   : Fetches the next row in the result set.

   PDO::FETCH_ORI_PRIOR
   : Fetches the previous row in the result set.

   PDO::FETCH_ORI_FIRST
   : Fetches the first row in the result set.

   PDO::FETCH_ORI_LAST
   : Fetches the last row in the result set.

   PDO::FETCH_ORI_ABS
   : Fetches the absolute row in the result set. Requires a positive integer as the third argument to `PDOStatement::fetch()`.

PDO::FETCH_ORI_REL
Fetches the relative row in the result set. Requires a positive or negative integer as the third argument to `PDOStatement::fetch()`.

2. An integer requesting the absolute or relative row in the result set, corresponding to the fetch orientation requested in the second argument to `PDOStatement::fetch()`.

`PDOStatement::fetch()` returns `FALSE` when the last row in the result set has been retrieved for a forward-only result set.

```
$stmt = $conn->query("SELECT firstnme, lastname FROM employee");
while ($row = $stmt->fetch(PDO::FETCH_NUM)) {
  print "Name: <p>{$row[0] $row[1]}</p>";
}
```

Fetching large objects in PHP (PDO)

The PDO extension supports the entire range of DB2 data types, including character large object (CLOB) and binary large object (BLOB) data types. When you fetch a large object from a result set, you can treat the large object simply as a PHP string. However, treating a large object as a PHP string is an approach that consumes more resources on your PHP server than necessary. If your ultimate goal is to create a file that contains the data for a large object, you can save system resources by fetching large objects directly into a file on your PHP server.

You must set up the PHP 5.1 or higher environment on your system and enable the PDO and PDO_IBM extensions.

To fetch a large object from the database directly into a file:

1. Create a PHP variable representing a stream–for example, the return value from a call to `fopen()`.
2. Call `PDO::prepare()` to create a PDOStatement object from an SQL statement.
3. Call `PDOStatement::bindColumn()` to bind the output column for the large object to the PHP variable representing the stream. The third parameter is a string representing the name of the PHP variable that holds the name of the file that is to contain the data from the large object. The fourth parameter is a PHP constant, `PDO::PARAM_LOB`, which tells the PDO extension to write the data into a file. Note that you must call `PDOStatement::bindColumn()` to assign a different PHP variable for every column in the result set.
4. Call `PDOStatement::execute()` to issue the SQL statement.
5. Call `PDOStatement::fetch(PDO::FETCH_BOUND)` to retrieve the next row in the result set, binding the column output into the PHP variables you associated with the `PDOStatement::bindColumn()` method.

```
$stmt = $conn->prepare("SELECT name, picture FROM animal_pictures");
$picture = fopen("/opt/albums/spook/grooming.jpg", "wb");
$stmt->bindColumn($stmt, 1, $nickname, PDO::PARAM_STR, 32);
$stmt->bindColumn($stmt, 2, $picture, PDO::PARAM_LOB);
$stmt->execute();
$stmt->fetch(PDO::FETCH_BOUND);
```

Managing transactions in PHP (PDO)

By default, PDO opens every connection in autocommit mode. Autocommit mode helps prevent locking escalation issues that can impede the performance of highly scalable Web applications. In some scripts, however, you might need to roll back a transaction containing one or more SQL statements. PDO enables you to exert fine-grained control over your transactions.

You must set up the PHP environment on your system and enable the PDO extension.

To begin a transaction:

1. Call `PDO::beginTransaction()` to begin a new transaction.
2. Issue one or more SQL statements within the scope of the database transaction using the connection resource for which transactions have been enabled.
3. Commit or rollback the transaction:
   - To commit the transaction, call `PDO::commit()`.
   - To rollback the transaction, call `PDO::rollBack()`.

After you commit or rollback the transaction, PDO automatically resets the database connection to autocommit mode. If you issue SQL statements in a transaction and the script ends without explicitly committing or rolling back the transaction, PDO automatically rolls back any work performed in the transaction.

```
$conn = new PDO('ibm:SAMPLE', 'db2inst1', 'ibmdb2', array(
  PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
  // PDO::ERRMODE_EXCEPTION means an SQL error throws an exception
try {
  // Issue these SQL statements in a transaction within a try{} block
  $conn->beginTransaction();

  // One or more SQL statements

  $conn->commit();
}
catch (Exception $e) {
  // If something raised an exception in our transaction block of statements,
  // roll back any work performed in the transaction
  print '<p>Unable to complete transaction!</p>';
  $conn->rollBack();
}
```

Handling errors and warnings in PHP (PDO)

Problems occasionally happen when you attempt to connect to a database or issue an SQL statement. The password for your connection might be incorrect, the table you referred to in a SELECT statement might not exist, or the syntax for an SQL statement might be invalid. You need to code defensively and use the error-handling functions offered by PDO to enable your application to recover gracefully from a problem.

You must set up the PHP environment on your system and enable the PDO and PDO_IBM extensions.

PDO gives you the option of handling errors as warnings, errors, or exceptions. However, when you create a new PDO connection object, PDO always throws a `PDOException` object if an error occurs. If you do not catch the exception, PHP

prints a backtrace of the error information which might expose your database connection credentials, including your user name and password.

- To catch a `PDOException` object and handle the associated error:
  1. Wrap the call to the PDO constructor in a `try` block.
  2. Following the `try` block, include a `catch` block that catches the `PDOException` object.
  3. Retrieve the error message associated with the error by invoking the `Exception::getMessage()` method on the `PDOException` object.
- To retrieve the SQLSTATE associated with a `PDO` or `PDOStatement` object, invoke the `errorCode()` method on the object.
- To retrieve an array of error information associated with a `PDO` or `PDOStatement` object, invoke the `errorInfo()` method on the object. The array contains a string representing the SQLSTATE as the first element, an integer representing the SQL or CLI error code as the second element, and a string containing the full text error message as the third element.

Calling stored procedures

Calling stored procedures with OUT or INOUT parameters in PHP (PDO)

DB2 supports stored procedures with parameters that only accept an input value (IN parameters), that only return an output value (OUT parameters), or that accept an input value and return an output value (INOUT). With the PDO_IBM extension for PHP you can handle IN parameters like any other parameter marker in an SQL statement. However, the PDO_IBM extension also enables you to CALL stored procedures with OUT and INOUT parameters and retrieve the output values from those parameters.

You must set up the PHP environment on your system and enable the PDO and PDO_IBM extensions.

To call a stored procedure with OUT or INOUT parameters:

1. Call `PDO::prepare()` to prepare a CALL statement with parameter markers representing the OUT and INOUT parameters.
2. Call `PDOStatement::bindParam()` to bind each parameter marker to the name of the PHP variable that will hold the output value of the parameter after the CALL statement has been issued. For INOUT parameters, the value of the PHP variable is passed as the input value of the parameter when the CALL statement is issued. Set the third parameter for `PDOStatement::bindParam()` to the type of data being bound:

   PDO::PARAM_NULL
   Represents the SQL NULL data type.

   PDO::PARAM_INT
   Represents SQL integer types.

   PDO::PARAM_LOB
   Represents SQL large object types.

   PDO::PARAM_STR
   Represents SQL character data types.

3. For an INOUT parameter, use the bitwise OR operator to append `PDO::PARAM_INPUT_OUTPUT` to the type of data being bound.
4. Set the fourth parameter of `PDOStatement::bindParam()` to the maximum expected length of the output value.

```
$sql = 'CALL match_animal(?, ?)';
$stmt = $conn->prepare($sql);

$second_name = "Rickety Ride";
$weight = 0;

$stmt->bindParam(1, $second_name, PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT, 32);
$stmt->bindParam(2, $weight, PDO::PARAM_INT, 10);

print "Values of bound parameters _before_ CALL:\n";
print "  1: {$second_name} 2: {$weight}\n";

$stmt->execute();

print "Values of bound parameters _after_ CALL:\n";
print "  1: {$second_name} 2: {$weight}\n";
```

Calling stored procedures that return multiple result sets in PHP (PDO)

DB2 enables you to create and call stored procedures that return more than one result set. The PDO_IBM extension for PHP supports this capability through the `nextRowset()` method. You can use this method to fetch rows from different result sets returned by a single call to the same stored procedure.

- You must set up the PHP 5.1 or higher environment on your system and enable the PDO_IBM extension.
- You must have a PDOStatement object returned from calling a stored procedure with `PDO::query()` or `PDOStatement::execute()`.

To return multiple result sets from a stored procedure:

1. The first result set is associated with the PDOStatement object returned by the CALL statement. You can fetch rows from the PDOStatement object until no more rows are available in the first result set.
2. Call the `nextRowset()` method of the PDOStatement object to return the next result set. You can fetch rows from the PDOStatement object until no more rows are available in the next result set.

```
$sql = 'CALL multiple_results()';
$stmt = $conn->query($sql);
do {
   $rows = $stmt->fetchAll(PDO::FETCH_NUM);
   if ($rows) {
       print_r($rows);
   }
} while ($stmt->nextRowset());
```

Part 2. Developing Perl Applications

Chapter 5. Programming Considerations for Perl

Perl is a popular programming language that is freely available for many operating systems. Using the DBD::DB2 driver available from http://www.ibm.com/software/data/db2/perl with the Perl Database Interface DBI) Module available from http://www.perl.com, you can create DB2 applications using Perl.

Because Perl is an interpreted language and the Perl DBI Module uses dynamic SQL, Perl is an ideal language for quickly creating and revising prototypes of DB2 applications. The Perl DBI Module uses an interface that is quite similar to the CLI and JDBC interfaces, which makes it easy for you to port your Perl prototypes to CLI and JDBC.

Most database vendors provide a database driver for the Perl DBI Module, which means that you can also use Perl to create applications that access data from many different database servers. For example, you can write a Perl DB2 application that connects to an Oracle database using the DBD::Oracle database driver, fetch data from the Oracle database, and insert the data into a DB2 database using the DBD::DB2 database driver.

Database Connections in Perl

To enable Perl to load the DBI module, you must include the following line in your DB2 application:

```
use DBI;
```

The DBI module automatically loads the DBD::DB2 driver when you create a *database handle* using the DBI->connect statement with the following syntax:

```
my $dbhandle = DBI->connect'dbi:DB2:dbalias', $userID, $password);
```

where:

$dbhandle
represents the database handle returned by the connect statement

dbalias
represents a DB2 alias cataloged in your DB2 database directory

$userID
represents the user ID used to connect to the database

$password
represents the password for the user ID used to connect to the database

Fetching Results in Perl

Because the Perl DBI Module only supports dynamic SQL, you cannot use host variables in your Perl DB2 applications.

To return results from an SQL query, perform the following steps:

1. Create a database handle by connecting to the database with the `DBI->connect` statement.

2. Create a statement handle from the database handle. For example, you can call prepare with an SQL statement as a string argument to return statement handle *$sth* from the database handle, as demonstrated in the following Perl statement:

```
my $sth = $dbhandle->prepare
   'SELECT firstnme, lastname
      FROM employee '
   );
```

3. Execute the SQL statement by calling execute on the statement handle. A successful call to execute associates a result set with the statement handle. For example, you can execute the statement prepared in the previous example using the following Perl statement:

```
#Note: $rc represents the return code for the execute call
my $rc = $sth->execute);
```

4. Fetch a row from the result set associated with the statement handle with a call to fetchrow). The Perl DBI returns a row as an array with one value per column. For example, you can return all of the rows from the statement handle in the previous example using the following Perl statement:

```
while $firstnme, $lastname) = $sth->fetchrow))
   print "$firstnme $lastname\n";
}
```

Parameter Markers in Perl

To enable you to execute a prepared statement using different input values for specified fields, the Perl DBI module enables you to prepare and execute a statement using parameter markers. To include a parameter marker in an SQL statement, use the question mark ?) character.

The following Perl code creates a statement handle that accepts a parameter marker for the WHERE clause of a SELECT statement. The code then executes the statement twice using the input values 25000 and 35000 to replace the parameter marker.

```
my $sth = $dbhandle->prepare
   'SELECT firstnme, lastname
      FROM employee
      WHERE salary > ?'
   );

my $rc = $sth->execute25000);

⋮

my $rc = $sth->execute35000);
```

SQLSTATE and SQLCODE Variables in Perl

To return the SQLSTATE associated with a Perl DBI database handle or statement handle, call the state method. For example, to return the SQLSTATE associated with the database handle $dbhandle, include the following Perl statement in your application:

```
my $sqlstate = $dbhandle->state;
```

To return the SQLCODE associated with a Perl DBI database handle or statement handle, call the err method. To return the message for an SQLCODE associated with a Perl DBI database handle or statement handle, call the errstr method. For example, to return the SQLCODE associated with the database handle $dbhandle, include the following Perl statement in your application:

```
my $sqlcode = $dbhandle->err;
```

Perl Restrictions

The Perl DBI module supports only dynamic SQL. When you need to execute a statement multiple times, you can improve the performance of your Perl DB2 applications by issuing a prepare call to prepare the statement.

Perl does not support multiple-thread database access.

For current information on the restrictions of the version of the DBD::DB2 driver that you install on your workstation, refer to the CAVEATS file in the DBD::DB2 driver package.

Example of a Perl Program

Following is an example of an application written in Perl:

```
#!/usr/bin/perl
use DBI;

my $database='dbi:DB2:sample';
my $user='';
my $password='';

my $dbh = DBI->connect$database, $user, $password)
   or die "Can't connect to $database: $DBI::errstr";

my $sth = $dbh->prepare
   q SELECT firstnme, lastname
      FROM employee }
   )
   or die "Can't prepare statement: $DBI::errstr";

my $rc = $sth->execute
   or die "Can't execute statement: $DBI::errstr";

print "Query will return $sth->NUM_OF_FIELDS} fields.\n\n";
print "$sth->NAME}->0]: $sth->NAME}->1]\n";

while $firstnme, $lastname) = $sth->fetchrow))
   print "$firstnme: $lastname\n";
}

# check for problems which may have terminated the fetch early
warn $DBI::errstr if $DBI::err;

$sth->finish;
$dbh->disconnect;
```

Building Perl applications

DB2 supports database access for client applications written in Perl 5.8 or later. At the time of printing, release 1.0 of the DB2 driver (DBD::DB2) for the Perl Database Interface (Perl DBI) Version 1.41 or later is supported and available for AIX, HP-UX, Linux, Solaris and Windows. For information on how to obtain the latest driver, visit http://www.ibm.com/db2/perl.

DB2 provides Perl sample programs located on UNIX in the sqllib/samples/perl directory, and on Windows in the sqllib\samples\perl directory.

To run the perl interpreter on a Perl program on the command line, enter the interpreter name and the program name (including extension):

1. If connecting locally on the server:

   ```
   perl dbauth.pl
   ```

2. If connecting from a remote client:

   ```
   perl dbauth.pl sample <userid> <password>
   ```

Some programs require support files to be run. The `tbsel` sample program requires several tables created by the `tbselcreate.db2` CLP script. The `tbselinit` script (UNIX), or the `tbselinit.bat` batch file (Windows), first calls `tbseldrop.db2` to drop the tables if they exist, and then calls `tbselcreate.db2` to create them. So to run the program, you would enter the following commands:

1. If connecting locally on the server:

   ```
   tbselinit
   perl tbsel.pl
   ```

2. If connecting from a remote client:

   ```
   tbselinit
   perl tbsel.pl sample <userid> <password>
   ```

Note: For a remote client, you need to modify the connect statement in the `tbselinit` or `tbselinit.bat` file to hardcode your user ID and password: `db2 connect to sample user <userid> using <password>`

Part 3. Appendixes

Appendix A. Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics (Task, concept and reference topics)
  - Help for DB2 tools
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF DVD)
  - printed books
- Command line help
  - Command help
  - Message help

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hard-copy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com®.

You can access additional DB2 technical information such as technotes, white papers, and IBM Redbooks® publications online at ibm.com. Access the DB2 Information Management software library site at http://www.ibm.com/software/data/sw-library/.

Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how to improve the DB2 documentation, send an email to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this email address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/shop/publications/order. English DB2 Version 9.5 manuals in PDF format and translated versions can be downloaded from www.ibm.com/support/docview.wss?rs=71&uid=swg2700947.

Although the tables identify books available in print, the books might not be available in your country or region.

Table 1. DB2 technical information

Name	Form Number	Available in print
Administrative API Reference	SC23-5842-00	Yes
Administrative Routines and Views	SC23-5843-00	No
Call Level Interface Guide and Reference, Volume 1	SC23-5844-00	Yes
Call Level Interface Guide and Reference, Volume 2	SC23-5845-00	Yes
Command Reference	SC23-5846-00	Yes
Data Movement Utilities Guide and Reference	SC23-5847-00	Yes
Data Recovery and High Availability Guide and Reference	SC23-5848-00	Yes
Data Servers, Databases, and Database Objects Guide	SC23-5849-00	Yes
Database Security Guide	SC23-5850-00	Yes
Developing ADO.NET and OLE DB Applications	SC23-5851-00	Yes
Developing Embedded SQL Applications	SC23-5852-00	Yes
Developing Java Applications	SC23-5853-00	Yes
Developing Perl and PHP Applications	SC23-5854-00	No
Developing User-defined Routines (SQL and External)	SC23-5855-00	Yes
Getting Started with Database Application Development	GC23-5856-00	Yes
Getting Started with DB2 installation and administration on Linux and Windows	GC23-5857-00	Yes
Internationalization Guide	SC23-5858-00	Yes
Message Reference, Volume 1	GI11-7855-00	No
Message Reference, Volume 2	GI11-7856-00	No
Migration Guide	GC23-5859-00	Yes
Net Search Extender Administration and User's Guide **Note:** The content of this document is not included in the DB2 Information Center	SC23-8509-00	Yes
Partitioning and Clustering Guide	SC23-5860-00	Yes
Query Patroller Administration and User's Guide	SC23-8507-00	Yes
Quick Beginnings for IBM Data Server Clients	GC23-5863-00	No
Quick Beginnings for DB2 Servers	GC23-5864-00	Yes

Table 1. DB2 technical information (continued)

Name	Form Number	Available in print
Spatial Extender and Geodetic Data Management Feature User's Guide and Reference	SC23-8508-00	Yes
SQL Reference, Volume 1	SC23-5861-00	Yes
SQL Reference, Volume 2	SC23-5862-00	Yes
System Monitor Guide and Reference	SC23-5865-00	Yes
Text Search Guide	SC23-5866-00	Yes
Troubleshooting Guide	GI11-7857-00	No
Tuning Database Performance	SC23-5867-00	Yes
Visual Explain Tutorial	SC23-5868-00	No
What's New	SC23-5869-00	Yes
Workload Manager Guide and Reference	SC23-5870-00	Yes
pureXML Guide	SC23-5871-00	Yes
XQuery Reference	SC23-5872-00	No

Table 2. DB2 Connect-specific technical information

Name	Form Number	Available in print
Quick Beginnings for DB2 Connect Personal Edition	GC23-5839-00	Yes
Quick Beginnings for DB2 Connect Servers	GC23-5840-00	Yes
DB2 Connect User's Guide	SC23-5841-00	Yes

Table 3. Information Integration technical information

Name	Form Number	Available in print
Information Integration: Administration Guide for Federated Systems	SC19-1020-01	Yes
Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018-02	Yes
Information Integration: Configuration Guide for Federated Data Sources	SC19-1034-01	No
Information Integration: SQL Replication Guide and Reference	SC19-1030-01	Yes
Information Integration: Introduction to Replication and Event Publishing	SC19-1028-01	Yes

Ordering printed DB2 books

If you require printed DB2 books, you can buy them online in many but not all countries or regions. You can always order printed DB2 books from your local IBM representative. Keep in mind that some softcopy books on the *DB2 PDF Documentation* DVD are unavailable in print. For example, neither volume of the *DB2 Message Reference* is available as a printed book.

Printed versions of many of the DB2 books available on the DB2 PDF Documentation DVD can be ordered for a fee from IBM. Depending on where you are placing your order from, you may be able to order books online, from the IBM Publications Center. If online ordering is not available in your country or region, you can always order printed DB2 books from your local IBM representative. Note that not all books on the DB2 PDF Documentation DVD are available in print.

Note: The most up-to-date and complete DB2 documentation is maintained in the DB2 Information Center at http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

To order printed DB2 books:

- To find out whether you can order printed DB2 books online in your country or region, check the IBM Publications Center at http://www.ibm.com/shop/publications/order. You must select a country, region, or language to access publication ordering information and then follow the ordering instructions for your location.
- To order printed DB2 books from your local IBM representative:
  1. Locate the contact information for your local representative from one of the following Web sites:
     - The IBM directory of world wide contacts at www.ibm.com/planetwide
     - The IBM Publications Web site at http://www.ibm.com/shop/publications/order. You will need to select your country, region, or language to the access appropriate publications home page for your location. From this page, follow the "About this site" link.
  2. When you call, specify that you want to order a DB2 publication.
  3. Provide your representative with the titles and form numbers of the books that you want to order. For titles and form numbers, see "DB2 technical library in hardcopy or PDF format" on page 39.

Displaying SQL state help from the command line processor

DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

To invoke SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

For example, `? 08003` displays help for the 08003 SQL state, and `? 08` displays help for the 08 class code.

Accessing different versions of the DB2 Information Center

For DB2 Version 9.5 topics, the DB2 Information Center URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/

For DB2 Version 9 topics, the DB2 Information Center URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9/

For DB2 Version 8 topics, go to the Version 8 Information Center URL at: http://publib.boulder.ibm.com/infocenter/db2luw/v8/

Displaying topics in your preferred language in the DB2 Information Center

The DB2 Information Center attempts to display topics in the language specified in your browser preferences. If a topic has not been translated into your preferred language, the DB2 Information Center displays the topic in English.

- To display topics in your preferred language in the Internet Explorer browser:
  1. In Internet Explorer, click the **Tools** —> **Internet Options** —> **Languages...** button. The Language Preferences window opens.
  2. Ensure your preferred language is specified as the first entry in the list of languages.
     - To add a new language to the list, click the **Add...** button.

       Note: Adding a language does not guarantee that the computer has the fonts required to display the topics in the preferred language.
     - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
  3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.
- To display topics in your preferred language in a Firefox or Mozilla browser:
  1. Select the button in the **Languages** section of the **Tools** —> **Options** —> **Advanced** dialog. The Languages panel is displayed in the Preferences window.
  2. Ensure your preferred language is specified as the first entry in the list of languages.
     - To add a new language to the list, click the **Add...** button to select a language from the Add Languages window.
     - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
  3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

On some browser and operating system combinations, you might have to also change the regional settings of your operating system to the locale and language of your choice.

Updating the DB2 Information Center installed on your computer or intranet server

If you have installed the DB2 Information Center locally, you can download and install updates that IBM might make available.

Updating your locally-installed DB2 Information Center requires that you:

1. Stop the DB2 Information Center on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to download and apply updates.
2. Use the Update feature to see what updates are available. If there are updates that you would like to install, you can use the Update feature to download and install them

   Note: If your environment requires installing the DB2 Information Center updates on a machine that is not connected to the internet, you have to mirror the update site to a local file system using a machine that is connected to the internet and has the DB2 Information Center installed. If many users on your network will be installing the documentation updates, you can reduce the time required for individuals to perform the updates by also mirroring the update site locally and creating a proxy for the update site.
   If update packages are available, use the Update feature to download the packages. However, the Update feature is only available in stand-alone mode.
3. Stop the stand-alone Information Center, and restart the DB2 Information Center on your computer.

Note: On Windows Vista, the commands listed below must be run as an administrator. To launch a command prompt or graphical tool with full administrator privileges, right-click on the shortcut and then select **Run as administrator**.

To update the DB2 Information Center installed on your computer or intranet server:

1. Stop the DB2 Information Center.
   - On Windows, click **Start → Control Panel → Administrative Tools → Services**. Then right-click on **DB2 Information Center** service and select **Stop**.
   - On Linux, enter the following command:

     ```
     /etc/init.d/db2icdv95 stop
     ```
2. Start the Information Center in stand-alone mode.
   - On Windows:
     a. Open a command window.
     b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the <Program Files>\IBM\DB2 Information Center\Version 9.5 directory, where <Program Files> represents the location of the Program Files directory.
     c. Navigate from the installation directory to the doc\bin directory.
     d. Run the help_start.bat file:

        ```
        help_start.bat
        ```
   - On Linux:

a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the /opt/ibm/db2ic/V9.5 directory.
b. Navigate from the installation directory to the doc/bin directory.
c. Run the help_start script:

```
help_start
```

The systems default Web browser launches to display the stand-alone Information Center.

3. Click the Update button (). On the right hand panel of the Information Center, click Find Updates. A list of updates for existing documentation displays.
4. To initiate the download process, check the selections you want to download, then click Install Updates.
5. After the download and installation process has completed, click Finish.
6. Stop the stand-alone Information Center.
   - On Windows, navigate to the installation directory's doc\bin directory, and run the help_end.bat file:

     ```
     help_end.bat
     ```

     Note: The help_end batch file contains the commands required to safely terminate the processes that were started with the help_start batch file. Do not use `Ctrl-C` or any other method to terminate help_start.bat.
   - On Linux, navigate to the installation directory's doc/bin directory, and run the help_end script:

     ```
     help_end
     ```

     Note: The help_end script contains the commands required to safely terminate the processes that were started with the help_start script. Do not use any other method to terminate the help_start script.
7. Restart the DB2 Information Center.
   - On Windows, click **Start → Control Panel → Administrative Tools → Services**. Then right-click on **DB2 Information Center** service and select **Start**.
   - On Linux, enter the following command:

     ```
     /etc/init.d/db2icdv95 start
     ```

The updated DB2 Information Center displays the new and updated topics.

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 products. Lessons provide step-by-step instructions.

Before you begin

You can view the XHTML version of the tutorial from the Information Center at http://publib.boulder.ibm.com/infocenter/db2help/.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 tutorials

To view the tutorial, click on the title.

"pureXML™" in *pureXML Guide*
Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

"Visual Explain" in *Visual Explain Tutorial*
Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 products.

DB2 documentation
Troubleshooting information can be found in the DB2 Troubleshooting Guide or the Support and Troubleshooting section of the DB2 Information Center. There you will find information on how to isolate and identify problems using DB2 diagnostic tools and utilities, solutions to some of the most common problems, and other advice on how to solve problems you might encounter with your DB2 products.

DB2 Technical Support Web site
Refer to the DB2 Technical Support Web site if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the DB2 Technical Support Web site at http://www.ibm.com/software/data/db2/udb/support.html

Terms and Conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal use: You may reproduce these Publications for your personal, non commercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these Publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the Publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Appendix B. 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.

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

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country/region 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.

This document may provide links or references to non-IBM Web sites and resources. IBM makes no representations, warranties, or other commitments whatsoever about any non-IBM Web sites or third-party resources that may be referenced, accessible from, or linked from this document. A link to a non-IBM Web site does not mean that IBM endorses the content or use of such Web site or

its owner. In addition, IBM is not a party to or responsible for any transactions you may enter into with third parties, even if you learn of such parties (or use a link to such parties) from an IBM site. Accordingly, you acknowledge and agree that IBM is not responsible for the availability of such external sites or resources, and is not responsible or liable for any content, services, products, or other materials on or available from those sites or resources. Any software provided by third parties is subject to the terms and conditions of the license that accompanies that software.

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

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

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

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

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

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

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

Company, product, or service names identified in the documents of the DB2 Version 9.5 documentation library may be trademarks or service marks of International Business Machines Corporation or other companies. Information on the trademarks of IBM Corporation in the United States, other countries, or both is located at http://www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the DB2 documentation library:

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

Intel®, Intel logo, Intel Inside® logo, Intel Centrino®, Intel Centrino logo, Celeron®, Intel Xeon®, Intel SpeedStep®, Itanium® and Pentium® are trademarks of Intel Corporation in the United States, other countries, or both.

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

Adobe®, the Adobe logo, PostScript®, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

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

Index

A

application design
  Perl example 35
  prototyping in Perl 33
application development
  Perl
    building applications 35

D

databases
  connecting with Perl 33
DB2 Information Center
  updating 44
  versions 43
  viewing in different languages 43
documentation
  PDF or printed 39
  terms and conditions of use 46
documentation overview 39
dynamic SQL
  Perl support 33

E

error handling
  Perl 34
examples
  Perl program 35

H

help
  displaying 43
  for SQL statements 42
host variables
  unsupported in Perl 33

I

Information Center
  updating 44
  versions 43
  viewing in different languages 43

N

notices 49

O

ordering DB2 books 42

P

parameter markers
  Perl 34
Perl
  application example 35
  building applications 35
  connecting to database 33
  drivers 33
  parameter markers 34
  programming considerations 33
  restrictions 35
  returning data 33
  SQLCODEs 34
  SQLSTATEs 34
PHP 3
  ibm_db2
    Connecting to database 9
    Errors and warnings 18
    Executing SQL statements 12
    Fetching large objects 16
    Fetching result columns 15
    Fetching result rows 16
    Inserting large objects 14
    Preparing SQL statements 13
    Retrieving database metadata 9
    Stored procedure parameters 19
    Stored procedure results 20
    Transactions 17
    XQuery expressions 11
  Introduction 3
  pdo_ibm
    Connecting to database 21
    Errors and warnings 28
    Executing SQL with no result 22
    Executing SQL with result 22
    Fetching large objects 27
    Fetching result columns 25
    Fetching result rows 25
    Inserting large objects 24
    Preparing SQL statements 23
    Stored procedure parameters 29
    Stored procedure results 30
    Transactions 28
  Setup
    Linux 5
    Windows 5
printed books
  ordering 42
problem determination
  online information 46
  tutorials 46

R

retrieving data
  Perl 33

S

SQL statements
  displaying help 42
static SQL
  Perl, unsupported 35

T

terms and conditions
  use of publications 46
troubleshooting
  online information 46
  tutorials 46
tutorials
  troubleshooting and problem determination 46
  Visual Explain 45

U

updates
  DB2 Information Center 44
  Information Center 44

V

Visual Explain
  tutorial 45

Printed in USA

SC23-5854-00

Spine information:

DB2 Version 9.5 for Linux, UNIX, and Windows

Developing Perl and PHP Applications