IBM DB2 10.1
for Linux, UNIX, and Windows

Developing Perl, PHP, Python, and Ruby on Rails Applications

SC27-3876-00

IBM DB2 10.1
for Linux, UNIX, and Windows

Developing Perl, PHP, Python, and Ruby on Rails Applications

SC27-3876-00

Note

Before using this information and the product it supports, read the general information under Appendix B, "Notices," on page 79.

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

Chapter 1. Developing Perl Applications 1

Programming considerations for Perl 1
- Perl downloads and related resources 1
- Database connections in Perl 2
- Fetching results in Perl 3
- Parameter markers in Perl 4
- SQLSTATE and SQLCODE variables in Perl . . . 4
- Perl Restrictions 5
- pureXML and Perl 5
- Running Perl sample programs 7
- Executing routines from Perl applications. . . . 8

Chapter 2. Developing PHP applications 9

PHP application development for IBM data servers . 9
- PHP downloads and related resources 10
- Setting up the PHP environment 10
- Application development in PHP (ibm_db2) . . 14
- Application development in PHP (PDO). . . . 30

Chapter 3. Developing Python applications 43

Python, SQLAlchemy and Django Framework application development for IBM data servers. . . 43
- Python downloads and related resources . . . 43
- Setting up the Python environment for IBM data servers 44
- Application development in Python with ibm_db 47

Chapter 4. Developing Ruby on Rails applications 59

The IBM_DB Ruby driver and Rails adapter . . . 59
- Getting started with IBM data servers on Rails 59
- Installing the IBM_DB adapter and driver as a Ruby gem 60
- Configuring Rails application connections to IBM data servers 64
- IBM Ruby driver and trusted contexts 65
- IBM_DB Rails adapter dependencies and consequences 65
- The IBM_DB Ruby driver and Rails adapter are not supported on JRuby 66
- ActiveRecord-JDBC versus IBM_DB adapter . . 66
- Heap size considerations with DB2 on Rails . . 67

Appendix A. Overview of the DB2 technical information 69

DB2 technical library in hardcopy or PDF format . . 69
- Displaying SQL state help from the command line processor 72
- Accessing different versions of the DB2 Information Center 72
- Updating the DB2 Information Center installed on your computer or intranet server 72
- Manually updating the DB2 Information Center installed on your computer or intranet server . . . 74
- DB2 tutorials 75
- DB2 troubleshooting information 76
- Terms and conditions 76

Appendix B. Notices 79

Index 83

Chapter 1. Developing Perl Applications

Programming considerations for Perl

Perl Database Interface (DBI) is an open standard application programming interface (API) that provides database access for client applications written in Perl. Perl DBI defines a set of functions, variables, and conventions that provide a platform-independent database interface.

You can use the IBM® DB2® Database Driver for Perl DBI (the DBD::DB2 driver) available from http://www.ibm.com/software/data/db2/perl along with the Perl DBI Module available from http://www.perl.com to create DB2 applications that use 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.

For information about supported database servers, installation instructions, and prerequisites, see http://www.ibm.com/software/data/db2/perl

Perl downloads and related resources

Several resources are available to help you develop Perl applications that access IBM data servers.

Table 1. Perl downloads and related resources

Downloads	Related resources
Perl Database Interface (DBI) Module	http://www.perl.com
DBD::DB2 driver	http://www.ibm.com/software/data/db2/perl
IBM Data Server Driver Package (DS Driver)	http://www.ibm.com/software/data/support/data-server-clients/index.html
DBI API documentation	http://search.cpan.org/~timb/DBI/DBI.pm
DB2 Perl Database Interface for for Linux, UNIX, and Windows technote, including readme and installation instructions	http://www.ibm.com/software/data/db2/perl
Perl driver bug reporting system	http://rt.cpan.org/
Reporting bugs to the Open Source team at IBM	opendev@us.ibm.com

Database connections in Perl

The DBD::DB2 driver provides support for standard database connection functions defined by the DBI API.

To enable Perl to load the DBI module, you must include the `use DBI;` line in your application:

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

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

where:

$dbhandle
represents the database handle returned by the connect statement

dsn

for local connections, represents a DB2 alias cataloged in your DB2 database directory

for remote connections, represents a complete connection string that includes the host name, port number, protocol, user ID, and password for connecting to the remote host

$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

For more information about the DBI API, see http://search.cpan.org/~timb/DBI/DBI.pmhttp://search.cpan.org/~timb/DBI/DBI.pm.

Example

Example 1: Connect to a database on the local host (client and server are on the same workstation)

```
use DBI;

$DATABASE = 'dbname';
$USERID = 'username';
$PASSWORD = 'password';

my $dbh = DBI->connect("dbi:DB2:$DATABASE", $USERID, $PASSWORD, {PrintError => 0})
or die "Couldn't connect to database: " . DBI->errstr;

$dbh->disconnect;
```

Example 2: Connect to a database on the remote host (client and server are on different workstations)

```
use DBI;

$DSN="DATABASE=sample; HOSTNAME=host; PORT=60000; PROTOCOL=TCPIP; UID=username;
PWD=password";

my $dbh = DBI->connect("dbi:DB2:$DSN", $USERID, $PASSWORD, {PrintError => 0})
or die "Couldn't connect to database: " . DBI->errstr;

$dbh->disconnect;
```

Fetching results in Perl

The Perl DBI module provides methods for connecting to a database, preparing and issuing SQL statements, and fetching rows from result sets.

About this task

This procedure fetches results from an SQL query.

Restrictions

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

Procedure

To fetch results:

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 return the statement handle $sth from the database handle by calling the prepare method and passing an SQL statement as a string argument, as demonstrated in the Perl statement example:

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

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

   ```
   #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 by calling the fetchrow method. 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 by using the listed Perl statement:

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

Example

The example shows how to connect to a database and issue a SELECT statement from 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 that might have terminated the fetch early
warn $DBI::errstr if $DBI::err;

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

Parameter markers in Perl

The Perl DBI module supports executing a prepared statement that includes parameter markers for variable input. To include a parameter marker in an SQL statement, use the question mark (?) character or a colon followed by a name (`:name`).

The Perl code example 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->execute(25000);
```

•
•
•

```
my $rc = $sth->execute(35000);
```

SQLSTATE and SQLCODE variables in Perl

The Perl DBI module provides methods for returning the SQLSTATE and SQLCODE associated with a Perl DBI database or statement handle.

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 `my $sqlstate = $dbhandle->state;` Perl statement in your application:

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 `my $sqlcode = $dbhandle->err;` Perl statement in your application:

Perl Restrictions

Some restrictions apply to the support that is available for application development in Perl.

The Perl DBI module supports only dynamic SQL. When you must execute a statement multiple times, you can improve the performance of your Perl 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, see the CAVEATS file in the DBD::DB2 driver package.

pureXML and Perl

The DBD::DB2 driver supports DB2 pureXML®. Support for pureXML allows more direct access to your data through the DBD::DB2 driver and helps to decrease application logic by providing more transparent communication between your application and database.

With pureXML support, you can directly insert XML documents into your DB2 database. Your application no longer needs to parse XML documents because the pureXML parser is automatically run when you insert XML data into the database. Having document parsing handled outside your application improves application performance and reduces maintenance efforts. Retrieval of XML stored data with the DBD::DB2 driver is easy as well; you can access the data using a BLOB or record.

For information about the DB2 Perl Database Interface and how to download the latest DBD::DB2 driver, see http://www.ibm.com/software/data/db2/perl.

Example

The example is a Perl program that uses pureXML:

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

# Use DBD:DB2 module:
#   to create a simple DB2 table with an XML column
#   Add one row of data
#   retreive the XML data as a record or a LOB (based on $datatype).

# NOTE: the DB2 SAMPLE database must already exist.

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

my $datatype = "record" ;
#  $datatype = "LOB" ;

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

# For LOB datatype, LongReadLen = 0 -- no data is retrieved on initial fetch
$dbh->{LongReadLen} = 0 if $datatype eq "LOB" ;

# SQL CREATE TABLE to create test table
```

```
my $stmt = "CREATE TABLE xmlTest (id INTEGER, data XML)";
my $sth = $dbh->prepare($stmt);
$sth->execute();

#insert one row of data into table
insertData() ;

# SQL SELECT statement returns home phone element from XML data
$stmt = qq(
 SELECT XMLQUERY ('
 \$d/*:customerinfo/*:phone[\@type = "home"] '
 passing data as "d")
 FROM xmlTest
) ;


# prepare and execute SELECT statement
$sth = $dbh->prepare($stmt);
$sth->execute();

# Print data returned from select statement
if($datatype eq "LOB") {
    printLOB() ;
}
else {
 printRecord() ;
}

# Drop table
$stmt = "DROP TABLE xmlTest" ;
$sth = $dbh->prepare($stmt);
$sth->execute();

warn $DBI::errstr if $DBI::err;

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


##############


sub printRecord {
 print "output data as as record\n" ;

 while( my @row = $sth->fetchrow )
 {
  print $row[0] . "\n";
 }

 warn $DBI::errstr if $DBI::err;
}


sub printLOB {
 print "output as Blob data\n" ;

 my $offset = 0;
 my $buff="";
 $sth->fetch();
 while( $buff = $sth->blob_read(1,$offset,1000000)) {
  print $buff;
  $offset+=length($buff);
  $buff="";
 }
 warn $DBI::errstr if $DBI::err;
}
```

```
sub insertData {

 # insert a row of data
 my $xmlInfo = qq(\'
 <customerinfo xmlns="http://posample.org" Cid="1011">
   <name>Bill Jones</name>
   <addr country="Canada">
     <street>5 Redwood</street>
     <city>Toronto</city>
     <prov-state>Ontario</prov-state>
     <pcode-zip>M6W 1E9</pcode-zip>
   </addr>
   <phone type="work">416-555-9911</phone>
   <phone type="home">416-555-1212</phone>
 </customerinfo>
 \') ;

 my $catID = 1011 ;

 # SQL statement to insert data.
 my $Sql = qq(
  INSERT INTO xmlTest (id, data)
      VALUES($catID, $xmlInfo )
 );

 $sth = $dbh->prepare( $Sql )
   or die "Can't prepare statement: $DBI::errstr";

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

 # check for problems
 warn $DBI::errstr if $DBI::err;
}
```

Running Perl sample programs

Perl sample programs are available that demonstrate how to build a Perl application.

Before you begin

Before running the Perl sample programs, you must install the latest DB2::DB2 driver for Perl DBI. For information about how to obtain the latest driver, see http://www.ibm.com/software/data/db2/perl.

About this task

The Perl sample programs for DB2 database are available in the `sqllib/samples/perl` directory.

Procedure

To run the Perl interpreter on a Perl sample program on the command line:

Enter the interpreter name and the program name (including the file extension):

- If connecting locally on the server:

  ```
  perl dbauth.pl
  ```

- If connecting from a remote client:

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

Some of the sample programs require you to run support files. For example, the `tbsel` sample program requires several tables that are 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. Therefore, to run the `tbsel` sample program, issue the listed commands:

- If connecting locally on the server:

```
tbselinit
perl tbsel.pl
```

- If connecting from a remote client:

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

Note: For a remote client, you must 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>`

Executing routines from Perl applications

DB2 client applications can access routines (stored procedures and user-defined functions) that are created by supported host languages or by SQL procedures. For example, the sample program `spclient.pl` can access the SQL procedures spserver shared library, if it exists in the database.

Before you begin

To build a host language routine, you must have the appropriate compiler set up on the server. SQL procedures do not require a compiler. The shared library can be built on the server only, and not from a remote client.

Procedure

To create SQL procedures in a shared library and then accesses the procedures from a Perl application:

1. Create and catalog the SQL procedures in the library. For example, go to the `samples/sqlpl` directory on the server, and run the listed commands to create and catalog the SQL procedures in the `spserver` library:

```
db2 connect to sample
db2 -td@ -vf spserver.db2
```

2. Go back to the `perl` samples directory (this can be on a remote client workstation), and run the Perl interpreter on the client program to access the `spserver` shared library:
   - If connecting locally on the server:

```
perl spclient
```

   - If connecting from a remote client:

```
perl spclient sample <userid> <password>
```

Chapter 2. Developing PHP applications

PHP application development for IBM data servers

PHP: Hypertext Preprocessor (PHP) is an interpreted programming language that is widely used for developing web applications. PHP has become a popular language for web development because it is easy to learn, focuses on practical solutions, and supports the most commonly required functionality in web applications.

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 website.

IBM provides the listed PHP extensions for accessing IBM data server databases:

ibm_db2
: 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. This extension is written, maintained, and supported by IBM.

pdo_ibm
: A driver for the PHP Data Objects (PDO) extension that offers access to IBM data server databases through the standard object-oriented database interface introduced in PHP 5.1.

These extensions are included as part of the IBM Data Server Driver Package (DS Driver) of Version 1.7.0. This version or a later version is supported to connect to IBM DB2 Version 9.7 for Linux, UNIX, and Windows. You can check the version of ibm_db2 extension by issuing the **php --re ibm_db2** command.

The most recent versions of ibm_db2 and pdo_ibm are also available from the PHP Extension Community Library (PECL) at http://pecl.php.net/.

PHP applications can access the listed IBM data server databases:

- IBM DB2 Version 9.1 for Linux, UNIX, and Windows, Fix Pack 2 and later
- IBM DB2 Universal Database™ (DB2 UDB) Version 8 for Linux, UNIX, and Windows, Fixpak 15 and later
- Remote connections to IBM DB2 for IBM i V5R3
- Remote connections to IBM DB2 for IBM i Version 5.4 and later
- Remote connections to IBM DB2 for z/OS®, Version 8 and later

A third extension, Unified ODBC, has historically offered access to DB2 database systems. For new applications, however, you can use either ibm_db2 and pdo_ibm because they 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.

PHP downloads and related resources

Many resources are available to help you develop PHP applications for IBM data servers.

Table 2. PHP downloads and related resources

Downloads	
Complete PHP source code [1]	http://www.php.net/downloads.php
ibm_db2 and pdo_ibm from the PHP Extension Community Library (PECL)	http://pecl.php.net/
IBM Data Server Driver Package (DS Driver)	http://www.ibm.com/software/data/support/data-server-clients/index.html
Zend Server	http://www.zend.com/en/products/server/downloads
PHP Manual	http://www.php.net/docs.php
ibm_db2 API documentation	http://www.php.net/ibm_db2
PDO API documentation	http://php.net/manual/en/book.pdo.php
PHP website	http://www.php.net/

1. Includes Windows binaries. Most Linux distributions come with PHP already precompiled.

Setting up the PHP environment

You can set up the PHP environment on Linux, UNIX, or Windows operating systems by installing a precompiled binary version of PHP and enabling support for IBM data servers.

About this task

For the easiest installation and configuration experience on Linux, UNIX, or Windows operating systems, you can download and install Zend Server for use in production systems at http://www.zend.com/en/products/server/downloads. Packaging details are available at http://www.zend.com/en/products/server/editions.

On Windows, precompiled binary versions of PHP are available for download from http://www.php.net/downloads.php. 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.

For more information about installing and configuring PHP, see http://www.php.net/manual/en/install.php.

Setting up the PHP environment on Windows

Before you can connect to an IBM data server and execute SQL statements, you must set up the PHP environment.

Before you begin

You must have the required software installed on your system:

- An Apache HTTP Server
- One of the listed client types: IBM Data Server Driver Package, IBM Data Server Client, or IBM Data Server Driver for ODBC and CLI

About this task

This procedure manually installs a precompiled binary version of PHP and enables support for IBM data servers on Windows.

Procedure

To set up the PHP environment on Windows:

1. Download the latest version of the PHP compressed package (as of now PHP 5.3.x) from http://windows.php.net/download/ and the collection of PECL modules from http://php.net/releases/index.php. The PECL collection modules are available only for PHP versions 5.2.x. Please note that Non Thread Safe build of PHP is recommended when you are not using PHP as an Apache module.
2. Extract the PHP compressed package into an installation directory.
3. Extract the collection of PECL modules 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 lines in the code block.
   - 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. If you are using Apache HTTP Server 2.x., enable PHP support by adding the example lines to your `httpd.conf` file, in which *phpdir* refers to the PHP installation directory:

   ```
   LoadModule php5_module 'phpdir/php5apache2_2.dll'
   AddType application/x-httpd-php .php
   PHPIniDir 'phpdir'
   ```

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

Results

Note: If you encounter message DB21085I or SQL09054, one of the listed task must be performed:

- Rebuild PHP in 64 bit mode
- Set the *PHP_IBM_DB2_LIB* and *PHP_PDO_IBM_LIB* variables to use `lib32` instead of default `lib64`, and update *LD_LIBRARY_PATH* to point to `lib32`.

The PHP extensions are now installed on your system and ready to use.

What to do next

Connect to the data server and begin executing SQL statements.

Setting up the PHP environment on Linux or UNIX

Before you can connect to an IBM data server and execute SQL statements, you must set up the PHP environment.

Before you begin

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.

You must have the listed software and files installed on your system:

- The Apache HTTP Server
- The gcc compiler and apache-devel, autoconf, automake, bison, flex, gcc, and libxml2-devel packages.
- If you are connecting from PHP to a DB2 database server on a remote machine, then you need one of the DB2 clients on the machine where you are installing, running or executing PHP, such as IBM Data Server Driver Package, IBM Data Server Client, or IBM Data Server Driver for ODBC and CLI.
- If you are connecting from PHP to a DB2 server on the local machine, you don't need to install any DB2 client.

About this task

This procedure manually compiles and installs PHP from source with support for DB2 on Linux or UNIX.

Procedure

To set up the PHP environment 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 5.3.x.
2. Untar the file by issuing the `tar -xjf php-5.x.x.tar.bz2` command.
3. Change directories to 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 listed options:

   ```
   ./configure --enable-cli --disable-cgi --with-apxs2=/usr/sbin/apxs2
   --with-zlib --with-pdo-ibm=<sqllib> --with-IBM_DB2=<sqllib>
   ```

   The configure options have the listed 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 CLI library to access database systems. The *<sqllib>* setting refers to the directory in which DB2 is installed.

If the source code of pdo_ibm extensions are not in the ext/ directory under PHP source, this flag will not be valid. To use --with-pdo-ibm, you must have the pdo_ibm directory, containing source code of pdo_ibm in the ext/ subdirectory.

--with-IBM_DB2=<sqllib>
Enables the IBM_DB2 driver using the library to access database systems. The <*sqllib*> setting refers to the directory in which DB2 is installed.

If the source code of IBM_DB2 extension is not in the ext/ directory under PHP source, this flag will not be valid. To use this configure option, you must have the ibm_db2 directory, containing source code of ibm_db2 in the ext/ subdirectory.

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 installation directory using the configure command, you might need root authority to successfully issue this command. This installs the executable files and update the Apache HTTP Server configuration to support PHP.
7. Install the ibm_db2 extension by issuing the pecl install ibm_db2 command as a user with root authority.

   This command downloads, configures, compiles, and installs the ibm_db2 extension for PHP. It is recommended to use the latest extension. However, you can also use the ibm_db2 extension which is included as part of the DB2 products.
8. Copy the php.ini-development or php.ini-production 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 listed lines, where *instance* refers to the name of the DB2 instance on Linux or UNIX.
   - To enable the pdo_ibm extension and set the environment:

     ```
     extension=pdo_ibm.so
     PDO_IBM.db2_instance_name=instance
     ```
   - To enable the ibm_db2 extension and set the DB2 environment:

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

   Where the extension variable can be specified as a relative path from the extension directory, which is specified in extension_dir variable. For example, if the extension_dir is *$HOME*/usr/php/ext and the extension is at *$HOME*/user/sqllib/php32, the entry will appear as:extension=../../sqllib/php32/ibm_db2_5.2.1.so
10. Before you run any PHP script that connects to DB2, you must ensure that the IBM DB2 PHP drivers (PDO_IBM/IBM_DB2) can access the CLI driver libdb2.so, which is part of your DB2 Server setup or DB2 client setup. Failing to do so will result in a missing libraries - libdb2.so.1 error when you run your PHP script. On Linux, you do this by adding the folder where the libdb2.so file resides to the **LD_LIBRARY_PATH** environmental variable for the userid under which PHP is going to be run.

    When using IBM Data Server Driver Package, libdb2.so is located in the odbc_cli_driver/linux/clidriver/lib directory.

    In the DB2 server installation, libdb2.so is located in the sqllib/lib directory.
11. Restart the Apache HTTP Server to enable the changed configuration.

Results

Note: If you encounter message DB21085I or SQL09054, you can perform one of the listed tasks:

- Rebuild PHP in 64 bit mode
- Set the *PHP_IBM_DB2_LIB* and *PHP_PDO_IBM_LIB* variables to use `lib32` instead of default `lib64`, and update *LD_LIBRARY_PATH* to point to `lib32`.

Application development in PHP (ibm_db2)

The ibm_db2 extension provides a variety of useful PHP functions for accessing and manipulating data in an IBM data server database. The extension includes functions for connecting to a database, executing and preparing SQL statements, fetching rows from result sets, calling stored procedures, handling errors, and retrieving metadata.

Connecting to an IBM data server database in PHP (ibm_db2)

Before you can issue SQL statements to create, update, delete, or retrieve data, you must connect to a database from your PHP application. You can use the ibm_db2 API to connect to an IBM data server database through either a cataloged connection or a direct TCP/IP connection. To improve performance, you can also create a persistent connection.

Before you begin

Before connecting to an IBM data server database through the ibm_db2 extension, you must set up the PHP environment on your system and enable the ibm_db2 extension.

Procedure

To return a connection resource that you can use to call SQL statements, call one of the listed connection functions:

Table 3. ibm_db2 connection functions

Function	Description
db2_connect	Creates a non-persistent connection.
db2_pconnect	Creates a persistent connection. A persistent connection remains open between PHP requests, which allows subsequent PHP script requests to reuse the connection if they have an identical set of credentials.

The database values that you pass as arguments to these functions can specify either a cataloged database name or a complete database connection string for a direct TCP/IP connection. You can specify optional arguments that control when transactions are committed, the case of the column names that are returned, and the cursor type.
If the connection attempt fails, you can retrieve diagnostic information by calling the db2_conn_error or db2_stmt_errormsg function.
When you create a connection by calling the db2_connect function, PHP closes the connection to the database when one of the listed events occurs:

- You call the db2_close function for the connection
- You set the connection resource to NULL
- The PHP script finishes

When you create a connection by calling the db2_pconnect function, PHP ignores any calls to the db2_close function for the specified connection resource, and keeps the connection to the database open for subsequent PHP scripts.
For more information about the ibm_db2 API, see http://www.php.net/docs.php.

Example

Connect to a cataloged database.

```
<?php
$database = "sample";
$user = "db2inst1";
$password = "";

$conn = db2_connect($database, $user, $password);

if ($conn) {
echo "Connection succeeded.";
db2_close($conn);
}
else {
echo "Connection failed.";
}
?>
```

What to do next

If the connection attempt is successful, you can use the connection resource when you call ibm_db2 functions that execute SQL statements. Next, prepare and execute SQL statements.

Trusted contexts in PHP applications (ibm_db2):

Starting in Version 9.5 Fix Pack 3 (or later), the ibm_db2 extension supports trusted contexts by using connection string keywords.

Trusted contexts provide a way of building much faster and more secure three-tier applications. The user's identity is always preserved for auditing and security purposes. When you need secure connections, trusted contexts improve performance because you do not have to get new connections.

Example

Enable trusted contexts, switch users, and get the current user ID.

```
<?php

$database = "SAMPLE";
$hostname = "localhost";
$port = 50000;
$authID = "db2inst1";
$auth_pass = "ibmdb2";

$tc_user = "tcuser";
$tc_pass = "tcpassword";

$dsn = "DATABASE=$database;HOSTNAME=$hostname;PORT=$port;PROTOCOL=TCPIP;UID=$authID;PWD=$auth_pass;";
$options = array ("trustedcontext" => DB2_TRUSTED_CONTEXT_ENABLE);

$tc_conn = db2_connect($dsn, "", "", $options);
if($tc_conn) {
 echo "Explicit Trusted Connection succeeded.\n";

 if(db2_get_option($tc_conn, "trustedcontext")) {
  $userBefore = db2_get_option($tc_conn, "trusted_user");

  //Do some work as user 1.

  //Switching to trusted user.
```

```
  $parameters = array("trusted_user" => $tc_user, "trusted_password" => $tcuser_pass);
  $res = db2_set_option ($tc_conn, $parameters, 1);

  $userAfter = db2_get_option($tc_conn, "trusted_user");
  //Do more work as trusted user.

  if($userBefore != $userAfter) {
   echo "User has been switched." . "\n";
  }
 }

 db2_close($tc_conn);
}
else {
 echo "Explicit Trusted Connection failed.\n";
}

?>
```

Executing SQL statements in PHP (ibm_db2)

After connecting to a database, use functions available in the ibm_db2 API to prepare and execute SQL statements. The SQL statements can contain static text, XQuery expressions, or parameter markers that represent variable input.

Executing a single SQL statement in PHP (ibm_db2):

To prepare and execute a single SQL statement that accepts no input parameters, use the db2_exec function. A typical use of the db2_exec function is to set the default schema for your application in a common include file or base class.

Before you begin

To avoid the security threat of SQL injection attacks, use the db2_exec function only 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.

Obtain a connection resource by calling one of the connection functions in the ibm_db2 API. Refer to “Connecting to an IBM data server database in PHP (ibm_db2)” on page 14.

Procedure

To prepare and execute a single SQL statement, call the db2_exec function, passing the listed arguments:

connection
A valid database connection resource returned from the db2_connect or db2_pconnect function.

statement
A string that contains the SQL statement. This string can include an XQuery expression that is called by the XMLQUERY function.

options
Optional: An associative array that specifies 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 the database. You can set this parameter to DB2_CASE_LOWER to force column names to lowercase, 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.

If the function call succeeds, it returns a statement resource that you can use in subsequent function calls related to this query.
If the function call fails (returns `False`), you can use the db2_stmt_error or db2_stmt_errormsg function to retrieve diagnostic information about the error.
For more information about the ibm_db2 API, see http://www.php.net/docs.php.

Example

Example 1: Executing a single SQL statement.

```
<?php
$conn = db2_connect("sample", "db2inst1", "");
$sql = "SELECT * FROM DEPT";
$stmt = db2_exec($conn, $sql);
db2_close($conn);
?>
```

Example 2: Executing an XQuery expression

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

What to do next

If the SQL statement selected rows using a scrollable cursor, or inserted, updated, or deleted rows, you can call the db2_num_rows function 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 with variable input in PHP (ibm_db2):

To prepare and execute an SQL statement that includes variable input, use the db2_prepare, db2_bind_param, and db2_execute functions. Preparing a statement improves performance because the database server creates an optimized access plan for data retrieval that it can reuse if the statement is executed again.

Before you begin

Obtain a connection resource by calling one of the connection functions in the ibm_db2 API. Refer to “Connecting to an IBM data server database in PHP (ibm_db2)” on page 14.

Procedure

To prepare and execute an SQL statement that includes parameter markers:

1. Call the db2_prepare function, passing the listed arguments:

connection
A valid database connection resource returned from the db2_connect or db2_pconnect function.

statement
A string that contains the SQL statement, including question marks (?) as parameter markers for any column or predicate values that require variable input. This string can include an XQuery expression that is called the XMLQUERY function. You can only use parameter markers as a place holder for column or predicate values. The SQL compiler is unable to create an access plan for a statement that uses parameter markers in place of column names, table names, or other SQL identifiers.

options
Optional: An associative array that specifies 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 the database. You can set this parameter to DB2_CASE_LOWER to force column names to lowercase, 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.

If the function call succeeds, it returns a statement handle resource that you can use in subsequent function calls that are related to this query.

If the function call fails (returns `False`), you can use the db2_stmt_error or db2_stmt_errormsg function to retrieve diagnostic information about the error.

2. Optional: For each parameter marker in the SQL string, call the db2_bind_param function, passing the listed arguments. Binding input values to parameter markers ensures that each input value is treated as a single parameter, which prevents SQL injection attacks against your application.

   stmt
   A prepared statement returned by the call to the db2_prepare function.

   parameter-number
   An integer that represents the position of the parameter marker in the SQL statement.

   variable-name
   A string that specifies the name of the PHP variable to bind to the parameter specified by *parameter-number*.

3. Call the db2_execute function, passing the listed arguments:

   stmt
   A prepared statement returned by the db2_prepare function.

parameters
Optional: An array that contains the values to use in place of the parameter markers, in order.

For more information about the ibm_db2 API, see http://www.php.net/docs.php.

Example

Prepare and execute a statement that includes variable input.

```
$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']));
```

What to do next

If the SQL statement returns one or more result sets, you can begin fetching rows from the statement resource.

Inserting large objects in PHP (ibm_db2):

When you insert a large object into the database, rather than loading all of the data for a large object into a PHP string and passing it to the IBM data server database through an INSERT statement, you can insert large objects directly from a file on your PHP server.

Before you begin

Obtain a connection resource by calling one of the connection functions in the ibm_db2 API.

Procedure

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

1. Call the db2_prepare function to prepare an INSERT statement with a parameter marker that represents 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 file.
3. Call the db2_bind_param function to bind the parameter marker to the variable. The third argument to this function is a string representing the name of the PHP variable that holds the path and name of the file. The fourth argument is DB2_PARAM_FILE, which tells the ibm_db2 extension to retrieve the data from a file.
4. Call the db2_execute function to issue the INSERT statement.

Example

Insert a large object 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 rows or columns from result sets in PHP (ibm_db2):

After executing a statement that returns one or more result sets, use one of the functions available in the ibm_db2 API to iterate through the returned rows of each result set. If your result set includes columns that contain extremely large data, you can retrieve the data on a column-by-column basis to avoid using too much memory.

Before you begin

You must have a statement resource returned by either the db2_exec or db2_execute function that has one or more associated result sets.

Procedure

To fetch data from a result set:

1. Fetch data from a result set by calling one of the fetch functions.

Table 4. ibm_db2 fetch functions

Function	Description
db2_fetch_array	Returns an array, indexed by column position, representing a row in a result set. The columns are 0-indexed.
db2_fetch_assoc	Returns an array, indexed by column name, representing a row in a result set.
db2_fetch_both	Returns an array, indexed by both column name and position, representing a row in a result set
db2_fetch_row	Sets the result set pointer to the next row or requested row. Use this function to iterate through a result set.
db2_fetch_object	Returns an object with properties representing columns in the fetched row. The properties of the object map to the names of the columns in the result set.

These functions accept the listed arguments:

stmt
A valid statement resource.

row_number
The number of the row that you want to retrieve from the result set. Row numbering begins with 1. Specify a value for this optional parameter if you

requested a scrollable cursor when you called the db2_exec or db2_prepare function. With the default forward-only cursor, each call to a fetch method returns the next row in the result set.

2. Optional: If you called the db2_fetch_row function, for each iteration over the result set, retrieve a value from the specified column by calling the db2_result function. You can specify the column by either passing an integer that represents the position of the column in the row (starting with 0), or a string that represents the name of column.
3. Continue fetching rows until the fetch function returns False, which indicates that you have reached the end of the result set.

   For more information about the ibm_db2 API, see http://www.php.net/docs.php.

Example

Example 1: Fetch rows from a result set by calling the db2_fetch_object function

```
<?php
$conn = db2_connect("sample", "db2inst1", "");
$sql = 'SELECT FIRSTNME, LASTNAME FROM EMPLOYEE WHERE EMPNO = ?';
$stmt = db2_prepare($conn, $sql);
db2_execute($stmt, array('000010'));
while ($row = db2_fetch_object($stmt)) {
  print "Name:
  {$row->FIRSTNME} {$row->LASTNAME}

  ";
  }
db2_close($conn);
?>
```

Example 2: Fetch rows from a result set by calling the db2_fetch_row function

```
<?php
$conn = db2_connect("sample", "db2inst1", "");
$sql = 'SELECT FIRSTNME, LASTNAME FROM EMPLOYEE WHERE EMPNO = ?';
$stmt = db2_prepare($conn, $sql);
db2_execute($stmt, array('000010'));
while (db2_fetch_row($stmt)) {
  $fname = db2_result($stmt, 0);
  $lname = db2_result($stmt, 'LASTNAME');
  print "
  Name: $fname $lname

  ";
  }
db2_close($conn);
?>
```

Example 3: Fetch rows from a result set by calling the db2_fetch_both function

```
<?php
$conn = db2_connect("sample", "db2inst1", "");
$sql = 'SELECT FIRSTNME, LASTNAME FROM EMPLOYEE WHERE EMPNO = ?';
$stmt = db2_prepare($conn, $sql);
db2_execute($stmt, array('000010'));
while ($row = db2_fetch_both($stmt)) {
  print "
  NAME: $row[0] $row[1]

  ";
  print "
  NAME: " . $row['FIRSTNME'] . " " . $row['LASTNAME'] . "
```

```
  ";
  }
db2_close($conn);
?>
```

What to do next

When you are ready to close the connection to the database, call the db2_close function. If you attempt to close a persistent connection that you created by using db2_pconnect, the close request returns TRUE, and the IBM data server client connection remains available for the next caller.

Fetching large objects in PHP (ibm_db2):

When you fetch a large object from a result set, rather than treating the large object as a PHP string, you can save system resources by fetching large objects directly into a file on your PHP server.

Before you begin

Obtain a connection resource by calling one of the connection functions in the ibm_db2 API.

Procedure

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

1. Create a PHP variable representing a stream. For example, assign the return value from a call to the fopen function to a variable.
2. Create a SELECT statement by calling the db2_prepare function.
3. Bind the output column for the large object to the PHP variable representing the stream by calling the db2_bind_param function. The third argument to this function is a string representing the name of the PHP variable that holds the path and name of the file. The fourth argument is DB2_PARAM_FILE, which tells the ibm_db2 extension to write the data into a file.
4. Issue the SQL statement by calling the db2_execute function.
5. Retrieve the next row in the result set by calling an ibm_db2 fetch function (for example, db2_fetch_object).

   For more information about the ibm_db2 API, see http://www.php.net/docs.php.

Example

Fetch a large object from the database.

```
$stmt = db2_prepare($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);
```

Calling stored procedures in PHP (ibm_db2)

To call a stored procedure from a PHP application, you prepare and execute an SQL CALL statement. The procedure that you call can include input parameters (IN), output parameters (OUT), and input and output parameters (INOUT).

Before you begin

Obtain a connection resource by calling one of the connection functions in the ibm_db2 API. Refer to "Connecting to an IBM data server database in PHP (ibm_db2)" on page 14.

Procedure

To call a stored procedure:

1. Call the db2_prepare function, passing the listed arguments:

   connection
   A valid database connection resource returned from db2_connect or db2_pconnect.

   statement
   A string that contains the SQL CALL statement, including parameter markers (?) for any input or output parameters

   options
   Optional: A associative array that specifies the type of cursor to return for result sets. You can use this parameter to request a scrollable cursor on database servers that support this type of cursor. By default, a forward-only cursor is returned.

2. For each parameter marker in the CALL statement, call the db2_bind_param function, passing the listed arguments:

   stmt
   The prepared statement returned by the call to the db2_prepare function.

   parameter-number
   An integer that represents the position of the parameter marker in the SQL statement.

   variable-name
   The name of the PHP variable to bind to the parameter specified by *parameter-number*.

   parameter-type
   A constant that specifies whether to bind the PHP variable to the SQL parameter as an input parameter (DB2_PARAM_INPUT), an output parameter (DB2_PARAM_OUTPUT), or a parameter that accepts input and returns output (DB2_PARAM_INPUT_OUTPUT).

   This step binds each parameter marker to the name of a PHP variable that will hold the output.

3. Call the db2_execute function, passing the prepared statement as an argument.

   For more information about the ibm_db2 API, see http://www.php.net/docs.php.

Example

Prepare and execute an SQL CALL statement.

```
$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";
```

What to do next

If the procedure call returns one or more result sets, you can begin fetching rows from the statement resource.

Retrieving multiple result sets from a stored procedure in PHP (ibm_db2):

When a single call to a stored procedure returns more than one result set, you can use the db2_next_result function of the ibm_db2 API to retrieve the result sets.

Before you begin

You must have a statement resource returned by the db2_exec or db2_execute function that has multiple result sets.

Procedure

To retrieve multiple result sets:

1. Fetch rows from the first result set returned from the procedure by calling one of the ibm_db2 fetch functions, passing the statement resource as an argument. (The first result set that is returned from the procedure is associated with the statement resource.)

Table 5. ibm_db2 fetch functions

Function	Description
db2_fetch_array	Returns an array, indexed by column position, representing a row in a result set. The columns are 0-indexed.
db2_fetch_assoc	Returns an array, indexed by column name, representing a row in a result set.
db2_fetch_both	Returns an array, indexed by both column name and position, representing a row in a result set
db2_fetch_row	Sets the result set pointer to the next row or requested row. Use this function to iterate through a result set.
db2_fetch_object	Returns an object with properties representing columns in the fetched row. The properties of the object map to the names of the columns in the result set.

2. Retrieve the subsequent result sets by passing the original statement resource as the first argument to the db2_next_result function. You can fetch rows from the statement resource until no more rows are available in the result set.

   The db2_next_result function returns False when no more result sets are available or if the procedure did not return a result set.

For more information about the ibm_db2 API, see http://www.php.net/docs.php.

Example

Retrieve multiple result sets from a stored procedure.

```
$stmt = db2_exec($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
  }
}
```

What to do next

When you are ready to close the connection to the database, call the db2_close function. If you attempt to close a persistent connection that you created by using db2_pconnect, the close request returns TRUE, and the persistent IBM data server client connection remains available for the next caller.

Commit modes in PHP applications (ibm_db2)

You can control how groups of SQL statements are committed by specifying a commit mode for a connection resource. The ibm_db2 extension supports two commit modes: autocommit and manual commit.

You must use a regular connection resource returned by the db2_connect function to control database transactions in PHP. Persistent connections always use autocommit mode.

autocommit mode

In autocommit mode, each SQL statement is a complete transaction, which is automatically committed. Autocommit mode helps prevent locking escalation issues that can impede the performance of highly scalable Web applications. By default, the ibm_db2 extension opens every connection in autocommit mode.

You can turn on autocommit mode after disabling it by calling `db2_autocommit($conn, DB2_AUTOCOMMIT_ON)`, where *conn* is a valid connection resource.

Calling the db2_autocommit function might affect the performance of your PHP scripts because it requires additional communication between PHP and the database management system.

manual commit mode

In manual commit mode, the transaction ends when you call the db2_commit or db2_rollback function. This means that all statements executed on the same connection between the start of a transaction and the call to the commit or rollback function are treated as a single transaction.

Manual commit mode is useful if you might have to roll back a transaction that contains one or more SQL statements. 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.

You can turn off autocommit mode when you create a database connection by using the "AUTOCOMMIT" => DB2_AUTOCOMMIT_OFF setting in the db2_connect options array. You can also turn off autocommit mode for an existing connection resource by calling `db2_autocommit($conn, DB2_AUTOCOMMIT_OFF)`, where *conn* is a valid connection resource.

For more information about the ibm_db2 API, see http://www.php.net/docs.php.

Example

End the transaction when db2_commit or db2_rollback is called.

```
$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);
}
```

Error-handling functions in PHP applications (ibm_db2)

Sometimes errors happen when you attempt to connect to a database or issue an SQL statement. The username or password might be incorrect, a table or column name might be misspelled, or the SQL statement might be invalid. The ibm_db2 API provides error-handling functions to help you recover gracefully from these situations.

Connection errors

Use one of the listed functions to retrieve diagnostic information if a connection attempt fails.

Table 6. ibm_db2 functions for handling connection errors

Function	Description
db2_conn_error	Retrieves the SQLSTATE returned by the last connection attempt
db2_conn_errormsg	Retrieves a descriptive error message appropriate for an application error log

SQL errors

Use one of the listed functions to retrieve diagnostic information if an attempt to prepare or execute an SQL statement or to fetch a result from a result set fails.

Table 7. ibm_db2 functions for handling SQL errors

Function	Description
db2_stmt_error	Retrieves the SQLSTATE returned by the last attempt to prepare or execute an SQL statement or to fetch a result from a result set
db2_stmt_errormsg	Retrieves a descriptive error message appropriate for an application error log

For more information about the ibm_db2 API, see http://www.php.net/docs.php.

Tip: To avoid security vulnerabilities that might result 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. For example:

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

Example

Example 1: Handle connection errors

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

Example 2: Handle SQL errors

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

Example 3: Handle SQL errors that result from executing prepared statements

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

Database metadata retrieval functions in PHP (ibm_db2)

You can use functions in the ibm_db2 API to retrieve metadata for databases served by DB2 Database for Linux, UNIX, and Windows, IBM Cloudscape, and, through DB2 Connect™, DB2 for z/OS and DB2 for i.

Some classes of applications, such as administration interfaces, must 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 might change between versions of DB2, or the schema of the system catalog tables on DB2 Database for Linux, UNIX, and Windows might differ from the schema of the system catalog tables on DB2 for z/OS. Rather than laboriously maintaining these differences in your application code, you can use PHP functions available in the ibm_db2 extension to retrieve database metadata.

Before calling these functions, you must set up the PHP environment and have a connection resource returned by the db2_connect or db2_pconnect function.

Important: Calling metadata functions uses a significant amount of space. If possible, cache the results of your calls for use in subsequent calls.

Table 8. ibm_db2 metadata retrieval functions

Function	Description
db2_client_info	Returns a read-only object with information about the IBM data server client
db2_column_privileges	Returns a result set listing the columns and associated privileges for a table
db2_columns	Returns a result set listing the columns and associated metadata for a table
db2_foreign_keys	Returns a result set listing the foreign keys for a table
db2_primary_keys	Returns a result set listing the primary keys for a table
db2_procedure_columns	Returns a result set listing the parameters for one or more stored procedures
db2_procedures	Returns a result set listing the stored procedures registered in the database
db2_server_info	Returns a read-only object with information about the database management system software and configuration
db2_special_columns	Returns a result set listing the unique row identifiers for a table
db2_statistics	Returns a result set listing the indexes and statistics for a table
db2_table_privileges	Returns a result set listing tables and their associated privileges in the database

Most of the ibm_db2 database metadata retrieval functions return result sets with columns defined for each function. To retrieve rows from the result sets, use the ibm_db2 functions that are available for this purpose.

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.

For more information about the ibm_db2 API, see http://www.php.net/docs.php.

Example

Example 1: Display a list of columns and associated privileges for a table

```
<?php
$conn = db2_connect('sample', 'db2inst1', 'ibmdb2');

if ($conn) {
 $stmt = db2_column_privileges($conn, NULL, NULL, 'DEPARTMENT');
 $row = db2_fetch_array($stmt);
 print $row[2] . "\n";
 print $row[3] . "\n";
 print $row[7];
 db2_close($conn);
}
else {
 echo db2_conn_errormsg();
 printf("Connection failed\n\n");
}
?>
```

Example 2: Display a list of primary keys for a table

```
<?php
$conn = db2_connect('sample', 'db2inst1', 'ibmdb2');

if ($conn) {
 $stmt = db2_primary_keys($conn, NULL, NULL, 'DEPARTMENT');
 while ($row = db2_fetch_array($stmt)) {
  echo "TABLE_NAME:\t" . $row[2] . "\n";
  echo "COLUMN_NAME:\t" . $row[3] . "\n";
  echo "KEY_SEQ:\t" . $row[4] . "\n";
 }

 db2_close($conn);
}
else {
 echo db2_conn_errormsg();
 printf("Connection failed\n\n");
}
?>
```

Example 3: Display a list of parameters for one or more stored procedures

```
<?php
$conn = db2_connect('sample', 'db2inst1', 'ibmdb2');

if ($conn) {
 $stmt = db2_procedures($conn, NULL, 'SYS%', '%%');

 $row = db2_fetch_assoc($stmt);
 var_dump($row);

 db2_close($conn);
}
else {
 echo "Connection failed.\n";
}
?>
```

Example 4: Display a list of the indexes and statistics for a table

```
<?php
$conn = db2_connect('sample', 'db2inst1', 'ibmdb2');

if ($conn) {
 echo "Test DEPARTMENT table:\n";
 $result = db2_statistics($conn, NULL, NULL, "EMPLOYEE", 1);
 while ($row = db2_fetch_assoc($result)) {
  var_dump($row);
 }

 echo "Test non-existent table:\n";
 $result = db2_statistics($conn,NULL,NULL,"NON_EXISTENT_TABLE",1);
 $row = db2_fetch_array($result);
 if ($row) {
  echo "Non-Empty\n";
 } else {
  echo "Empty\n";
 }

 db2_close($conn);
}
else {
 echo 'no connection: ' . db2_conn_errormsg();
}
?>
```

Example 5: Display a list of tables and their associated privileges in the database

```
<?php
$conn = db2_connect('sample', 'db2inst1', 'ibmdb2');

if ($conn) {
 $stmt = db2_table_privileges($conn, NULL, "%%", "DEPARTMENT");
 while ($row = db2_fetch_assoc($stmt)) {
  var_dump($row);
 }
 db2_close($conn);
}
else {
 echo db2_conn_errormsg();
 printf("Connection failed\n\n");
}
?>
```

Application development in PHP (PDO)

The PDO_IBM extension provides a variety of useful PHP functions for accessing and manipulating data through the standard object-oriented database interface introduced in PHP 5.1. The extension includes functions for connecting to a database, executing and preparing SQL statements, fetching rows from result sets, managing transactions, calling stored procedures, handling errors, and retrieving metadata.

Connecting to an IBM data server database with PHP (PDO)

Before you can issue SQL statements to create, update, delete, or retrieve data, you must connect to a database. You can use the PHP Data Objects (PDO) interface for PHP to connect to an IBM data server database through either a cataloged connection or a direct TCP/IP connection. To improve performance, you can also create a persistent connection.

Before you begin

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

About this task

This procedure returns a connection object to an IBM data server database. This connection stays open until you set the PDO object to NULL, or the PHP script finishes.

Procedure

To connect to an IBM data server database:

1. Create a connection to the 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, the PDO_IBM extension uses connection pooling to minimize connection resources and improve connection performance.
   - (Linux and UNIX): To create a persistent connection, pass `array(PDO::ATTR_PERSISTENT => TRUE)` as the *driver_options* (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.

   For more information about the PDO API, see http://php.net/manual/en/book.pdo.php.

Example

Connect to an IBM data server database over a persistent connection.

```
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());
}
```

What to do next

Next, you prepare and execute SQL statements.

Executing SQL statements in PHP (PDO)

After connecting to a database, use methods available in the PDO API to prepare and execute SQL statements. The SQL statements can contain static text or parameter markers that represent variable input.

Executing a single SQL statement in PHP (PDO):

To prepare and execute a single SQL statement that accepts no input parameters, use the PDO::exec or PDO::query method. Use the PDO::exec method to execute a statement that returns no result set. Use the PDO::query method to execute a statement that returns one or more result sets.

Before you begin

Important: To avoid the security threat of SQL injection attacks, use the PDO::exec or PDO::query method only 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.

Obtain a connection object by calling the PDO constructor.

Procedure

To prepare and execute a single SQL statement that accepts no input parameters, call one of the listed methods:

- To execute an SQL statement that returns no result set, call the PDO::exec method on the PDO connection object, passing in a string that contains the SQL statement. For example, a typical use of PDO::exec is to set the default schema for your application in a common include file or base class.

  If the SQL statement succeeds (successfully inserts, modifies, or deletes rows), the PDO::exec method returns an integer value representing the number of rows that were inserted, modified, or deleted.

  To determine if the PDO::exec method failed (returned FALSE or 0), use the === operator to strictly test the returned value against FALSE.
- To execute an SQL statement that returns one or more result sets, call the PDO::query method on the PDO connection object, passing in a string that contains the SQL statement. For example, you might want to call this method to execute a static SELECT statement.

  If the method call succeeds, it returns a PDOStatement resource that you can use in subsequent method calls.

If the method call fails (returns FALSE), you can use the PDO::errorCode and PDO::errorInfo method to retrieve diagnostic information about the error.
For more information about the PDO API, see http://php.net/manual/en/book.pdo.php.

Example

Example 1: Call the PDO::exec method to set the default schema for your application

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

Example 2: Call the PDO::query method to issue an SQL SELECT statement

```
$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>";
}
```

What to do next

If you called the PDO::query method to create a PDOStatement object, you can begin retrieving rows from the object by calling the PDOStatement::fetch or PDOStatement::fetchAll method.

Preparing and executing SQL statements in PHP (PDO):

To prepare and execute an SQL statement that includes variable input, use the PDO::prepare, PDOStatement::bindParam, and PDOStatement::execute methods. Preparing a statement improves performance because the database server creates an optimized access plan for data retrieval that it can reuse if the statement is executed again.

Before you begin

Obtain a connection object by calling the PDO constructor. Refer to "Connecting to an IBM data server database with PHP (PDO)" on page 30.

Procedure

To prepare and execute an SQL statement that includes parameter markers:

1. Call the PDO::prepare method, passing the listed arguments:

   statement
   : A string that contains the SQL statement, including question marks (?) or named variables (`:name`) as parameter markers for any column or predicate values that require variable input. You can only use parameter markers as a place holder for column or predicate values. The SQL compiler is unable to create an access plan for a statement that uses 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.

   driver_options
   : Optional: An array that contains 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.

   If the function call succeeds, it returns a PDOStatement object that you can use in subsequent method calls that are related to this query.

   If the function call fails (returns `False`), you can use the PDO::errorCode or PDO::errorInfo method to retrieve diagnostic information about the error.

2. Optional: For each parameter marker in the SQL string, call the PDOStatement::bindParam method, passing the listed arguments. Binding input values to parameter markers ensures that each input value is treated as a single parameter, which prevents SQL injection attacks against your application.

   parameter
   A parameter identifier. For question mark parameter markers (?), this is an integer that represents the 1-indexed position of the parameter in the SQL statement. For named parameter markers (`:name`), this is a string that represents the parameter name.

   variable
   The value to use in place of the parameter marker

3. Call the PDOStatement::execute method, optionally passing an array that contains 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.

   For more information about the PDO API, see http://php.net/manual/en/book.pdo.php.

Example

Prepare and execute a statement that includes variable input.

```
$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']));
```

What to do next

If the SQL statement returns one or more result sets, you can begin fetching rows from the statement resource by calling the PDOStatement::fetch or PDOStatement::fetchAll method.

Inserting large objects in PHP (PDO):

When you insert a large object into the database, rather than loading all of the data for a large object into a PHP string and passing it to the IBM data server database through an INSERT statement, you can insert large objects directly from a file on your PHP server.

Before you begin

Obtain a connection object by calling the PDO constructor.

Procedure

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

1. Call the PDO::prepare method to create a PDOStatement object from an INSERT statement with a parameter marker that represents the large object column.
2. Create a PHP variable that represents a stream (for example, by assigning the value returned by the `fopen` function to variable).
3. Call the PDOStatement::bindParam method, passing the listed arguments to bind the parameter marker to the PHP variable that represents the stream of data for the large object:

   parameter
   A parameter identifier. For question mark parameter markers (?), this is an integer that represents the 1-indexed position of the parameter in the SQL statement. For named parameter markers (`:name`), this is a string that represents the parameter name.

   variable
   The value to use in place of the parameter marker

   data_type
   The PHP constant, `PDO::PARAM_LOB`, which tells the PDO extension to retrieve the data from a file.
4. Call the PDOStatement::execute method to issue the INSERT statement.

Example

Insert a large object into the database.

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

Reading query result sets

Fetching rows or columns from result sets in PHP (PDO):

After executing a statement that returns one or more result sets, use one of the methods available in the PDO API to iterate through the returned rows. The PDO API also provides methods to fetch a single column from one or more rows in the result set.

Before you begin

You must have a statement resource returned by either the PDO::query or PDOStatement::execute method that has one or more associated result sets.

Procedure

To fetch data from a result set:

1. Fetch data from a result set by calling one of the fetch methods:
   - 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 the column name and 0-indexed column position in the row. To request a different return style, specify one of the `PDO::FETCH_*` constants as the first parameter when you call the PDOStatement::fetch method:

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.

If you requested a scrollable cursor when you called the PDO::query or PDOStatement::execute method, you can pass the listed optional parameters that control which rows are returned to the caller:

- One of the `PDO::FETCH_ORI_*` constants that represents the fetch orientation of the 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 the `PDOStatement::fetch` method.

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

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

2. Optional: Fetch a single column from one or more rows in a result set by calling one of the listed methods:
   - 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, the PDOStatement::fetchColumn returns the first column in the row.
   - To return an array that contains a single column from all of the remaining rows in the result set:

     Call the PDOStatement::fetchAll method, passing the PDO::FETCH_COLUMN constant as the first argument, and the column you want to retrieve as the second argument. Column numbers start at 0. If you do not specify a column, calling `PDOStatement::fetchAll(PDO::FETCH_COLUMN)` returns the first column in the row.

   For more information about the PDO API, see http://php.net/manual/en/book.pdo.php.

Example

Return an array indexed by column number.

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

What to do next

When you are ready to close the connection to the database, set the PDO object to NULL. The connection closes automatically when the PHP script finishes.

Fetching large objects in PHP (PDO):

When you fetch a large object from a result set, rather than treating the large object as a PHP string, you can save system resources by fetching large objects directly into a file on your PHP server.

Before you begin

Obtain a connection object by calling the PDO constructor.

Procedure

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

1. Create a PHP variable representing a stream. For example, assign the return value from a call to the fopen function to a variable.

2. Create a PDOStatement object from an SQL statement by calling the PDO::prepare method.
3. Bind the output column for the large object to the PHP variable representing the stream by calling the PDOStatement::bindColumn method. The second argument is a string representing the name of the PHP variable that holds the path and name of the file. The third argument is a PHP constant, PDO::PARAM_LOB, which tells the PDO extension to write the data into a file. You must call the PDOStatement::bindColumn method to assign a different PHP variable for every column in the result set.
4. Issue the SQL statement by calling the PDOStatement::execute method.
5. Call PDOStatement::fetch(PDO::FETCH_BOUND) to retrieve the next row in the result set, binding the column output to the PHP variables that you associated when you called the PDOStatement::bindColumn method.

Example

Fetch a large object from the database directly into a file.

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

Calling stored procedures in PHP (PDO)

To call a stored procedure from a PHP application, you execute an SQL CALL statement. The procedure that you call can include input parameters (IN), output parameters (OUT), and input and output parameters (INOUT).

Before you begin

Obtain a connection object by calling the PDO constructor.

About this task

This procedure prepares and executes an SQL CALL statement. For more information, also see the topic about preparing and executing SQL statements.

Procedure

To call a stored procedure:

1. Call the PDO::prepare method to prepare a CALL statement with parameter markers that represent the OUT and INOUT parameters.
2. For each parameter marker in the CALL statement, call the PDOStatement::bindParam method 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.
   a. Set the third parameter, *data_type*, to one of the PDO::PARAM_* constants that specifies 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.

For an INOUT parameter, use the bitwise OR operator to append PDO::PARAM_INPUT_OUTPUT to the type of data being bound.

b. Set the fourth parameter, *length*, to the maximum expected length of the output value.

3. Call the PDOStatement::execute method, passing the prepared statement as an argument.

   For more information about the PDO API, see http://php.net/manual/en/book.pdo.php.

Example

Prepare and execute an SQL CALL statement.

```
$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";
```

Retrieving multiple result sets from a stored procedure in PHP (PDO):

When a single call to a stored procedure returns more than one result set, you can use the PDOStatement::nextRow method of the PDO API to retrieve the result sets.

Before you begin

You must have a PDOStatement object returned by calling a stored procedure with the PDO::query or PDOStatement::execute method.

Procedure

To retrieve multiple result sets:

1. Fetch rows from the first result set returned from the procedure by calling one of the PDO fetch methods. (The first result set that is returned from the procedure is associated with the PDOStatement object returned by the CALL statement.)
   - 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.

Fetch rows from the PDOStatement object until no more rows are available in the first result set.

2. Retrieve the subsequent result sets by calling the PDOStatement::nextRowset method to return the next result set. You can fetch rows from the PDOStatement object until no more rows are available in the result set.

   The PDOStatement::nextRowset method returns False when no more result sets are available or the procedure did not return a result set.

   For more information about the PDO API, see http://php.net/manual/en/book.pdo.php.

Example

Retrieve multiple result sets from a stored procedure.

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

What to do next

When you are ready to close the connection to the database, set the PDO object to NULL. The connection closes automatically when the PHP script finishes.

Commit modes in PHP (PDO)

You can control how groups of SQL statements are committed by specifying a commit mode for a connection resource. The PDO extension supports two commit modes: autocommit and manual commit.

autocommit mode
: In autocommit mode, each SQL statement is a complete transaction, which is automatically committed. Autocommit mode helps prevent locking escalation issues that can impede the performance of highly scalable Web applications. By default, the PDO extension opens every connection in autocommit mode.

manual commit mode
: In manual commit mode, the transaction begins when you call the PDO::beginTransaction method, and it ends when you call either the PDO::commit or PDO::rollBack method. This means that any statements executed (on the same connection) between the start of a transaction and the call to the commit or rollback method are treated as a single transaction.

  Manual commit mode is useful if you might have to roll back a transaction that contains one or more SQL statements. 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.

  After you commit or rollback the transaction, PDO automatically resets the database connection to autocommit mode.

For more information about the PDO API, see http://php.net/manual/en/book.pdo.php.

Example

End the transaction when PDO::commit or PDO::rollBack is called.

```
$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)

Sometimes errors happen when you attempt to connect to a database or issue an SQL statement. The password for your connection might be incorrect, a table you referred to in a SELECT statement might not exist, or the SQL statement might be invalid. PDO provides error-handling methods to help you recover gracefully from these situations.

Before you begin

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

About this task

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 that might expose your database connection credentials, including your user name and password.

This procedure catches a PDOException object and handles the associated error.

Procedure

1. To catch a PDOException object and handle the associated error:
   a. Wrap the call to the PDO constructor in a `try` block.
   b. Following the `try` block, include a `catch` block that catches the PDOException object.
   c. Retrieve the error message associated with the error by invoking the Exception::getMessage method on the PDOException object.
2. To retrieve the SQLSTATE associated with a PDO or PDOStatement object, invoke the errorCode method on the object.
3. 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.

For more information about the PDO API, see http://php.net/manual/en/book.pdo.php.

Chapter 3. Developing Python applications

Python, SQLAlchemy and Django Framework application development for IBM data servers

Python is a general purpose, high level scripting language that is well suited for rapid application development. Python emphasizes code readability and supports a variety of programming paradigms, including procedural, object-oriented, aspect-oriented, meta, and functional programming. The Python language is managed by the Python Software Foundation.

The listed extensions are available for accessing IBM data server databases from a Python application:

ibm_db
This API is defined by IBM and provides the best support for advanced features. In addition to issuing SQL queries, calling stored procedures, and using pureXML, you can access metadata information.

ibm_db_dbi
This API implements Python Database API Specification v2.0. Because the ibm_db_dbi API conforms to the specification, it does not offer some of the advanced features that the ibm_db API supports. If you have an application with a driver that supports Python Database API Specification v2.0, you can easily switch to ibm_db. The ibm_db and ibm_db_dbi APIs are packaged together.

ibm_db_sa
This adaptor supports SQLAlchemy, which offers a flexible way to access IBM data servers. SQLAlchemy is a popular open source Python SQL toolkit and object-to-relational mapper (ORM).

ibm_db_django
This adaptor provides access to IBM data servers from Django. Django is a popular web framework used to build high-performing, elegant web applications quickly.

Python applications can access the listed IBM data servers:

- IBM DB2 Version 9.1 for Linux, UNIX, and Windows, Fix Pack 2 and later
- IBM DB2 Universal Database (DB2 UDB) Version 8 for Linux, UNIX, and Windows, Fixpak 15 and later
- Remote connections to IBM DB2 for IBM i V5R3, with PTF SI27358 (includes SI27250)
- Remote connections to IBM DB2 for IBM i 5.4 and later, with PTF SI27256
- Remote connections to IBM DB2 for z/OS, Version 8 and Version 9
- IBM Informix® Dynamic Server v11.10 and later

Python downloads and related resources

Many resources are available to help you develop Python applications for IBM data servers.

Table 9. Python downloads and related resources

Downloads	
Python (Includes Windows binaries. Most Linux distributions come with Python already precompiled.)	http://www.python.org/download/
SQLAlchemy	http://www.sqlalchemy.org/download.html
Django	http://www.djangoproject.com/download/
ibm_db and ibm_db_dbi extensions (including source code)	http://pypi.python.org/pypi/ibm_db/
	http://code.google.com/p/ibm-db/downloads/list
ibm_db_sa adapter for SQLAlchemy 0.4	http://code.google.com/p/ibm-db/downloads/list
	http://pypi.python.org/pypi/ibm_db_sa
ibm_db_django adaptor for Django 1.0.x and 1.1	http://code.google.com/p/ibm-db/downloads/list
	http://pypi.python.org/pypi/ibm_db_django
setuptools program	http://pypi.python.org/pypi/setuptools
IBM Data Server Driver Package (DS Driver)	http://www.ibm.com/software/data/support/data-server-clients/index.html
API documentation	
ibm_db API documentation	http://code.google.com/p/ibm-db/wiki/APIs
Python Database API Specification v2.0	http://www.python.org/dev/peps/pep-0249/
SQLAlchemy documentation	
Quick Getting Started Steps for ibm_db_sa	http://code.google.com/p/ibm-db/wiki/README
SQLAlchemy Documentation	http://www.sqlalchemy.org/docs/index.html
Django documentation	
Getting Started steps for ibm_db_django	http://code.google.com/p/ibm-db/wiki/ibm_db_django_README
Django Documentation	http://www.djangoproject.com
Additional resources	
Python Programming Language website	http://www.python.org/
The Python SQL Toolkit and Object Relational Mapper website	http://www.sqlalchemy.org/

Setting up the Python environment for IBM data servers

Before you can connect to an IBM data server and run SQL statements, you must set up the Python environment by installing the ibm_db (Python) and, optionally, the ibm_db_sa (SQLAlchemy) or ibm_db_django (Django) packages on your system.

Before you begin

You must have the listed software installed on your system:

- Python 2.5 or later, excluding Python 3.X. For Linux operating systems, you also require the python2.5-dev package.
- setuptools, a program available at http://pypi.python.org/pypi/setuptools. You can use this program to download, build, install, upgrade, and uninstall Python packages.
- If you are connecting from Python to a DB2 database server on a remote machine, then you need one of the listed DB2 clients on the machine where you are installing, running or executing Python: IBM Data Server Driver Package, IBM Data Server Client, or IBM Data Server Driver for ODBC and CLI.
- If you are connecting from Python to a DB2 database server on the local machine, you don't need to install a DB2 database client.

Procedure

To set up the Python environment:

1. Set up your Linux or Windows environment by using one of the listed approaches:

   To install the latest DB2 Python drivers and framework adaptors, obtain them from the community. The community versions always contain up-to-date fixes, the latest of which sometimes might not be available in the drivers/adapters included in the DB2 installation.

   - If you have Internet access, issue one of the listed commands:
     - To install ibm_db: `easy_install ibm_db`
     - To install both ibm_db_sa and ibm_db: `easy_install ibm_db_sa`
     - To install ibm_db_django: `easy_install ibm_db_django`

     This step installs the eggs under the `site-packages` directory where setuptools is installed.
   - If you do not have Internet access, copy the appropriate egg file for your system from http://code.google.com/p/ibm-db/downloads/list, and issue the `easy_install` command:

     ```
     easy_install egg_file_name
     ```

     where *egg_file_name* is the path to the egg file. For example, issue the `easy_install` command:

     ```
     easy_install /home/user/ibm_db-xx-py2.5-linux-i386.egg
     ```
   - To install from community source-code, download the source code from http://pypi.python.org/pypi/ibm_db/. Then build and install the driver. The instructions for building and installing the driver are in the `README` file that is included with the driver source code.
   - If you want to install the DB2 Python drivers and framework adaptors that are available in your DB2 image, go to the egg file's directory in the DB2 installation, for example: `/home/user/sqllib/python64/`. Then issue one of the listed commands:
     - To install ibm_db: `easy_install ibm_db_egg_file_name` for example; `easy_install ibm_db-xx-py2.5-linux-x86_64.egg`
     - To install ibm_db_sa: `easy_install ibm_db_sa_egg_file_name` for example; `easy_install ibm_db_sa-xx-py2.5.egg`
     - To install ibm_db_django: `easy_install ibm_db_django_egg_file_name` for example; `easy_install ibm_db_django-xx-py2.5.egg`

     This step installs the eggs under the `site-packages` directory where setuptools is installed.

2. Create an environment variable named **PYTHONPATH**, and set it to the path where you installed the ibm_db egg, as shown in the listed examples:
   - On Windows operating systems: `PYTHONPATH=setuptools_install_path\site-packages\ibm_db-xx.egg`
   - On Linux (BASH shell): `export PYTHONPATH=setuptools_install_path/site-packages/ibm_db-xx.egg`
3. Before you run any Python script that connects to DB2, you need to ensure that the IBM DB2 Python driver can access the CLI driver called `libdb2.so`, which is part of your DB2 Server setup or DB2 client setup. Failing to do so will result in a `'missing libraries - libdb2.so.1'` error when you run your Python program. On Linux, you must ensure that the IBM DB2 Python driver can access the CLI driver by adding the `libdb2.so` file path to the **LD_LIBRARY_PATH** environmental variable.

   When using IBM Data Server Driver Package, `libdb2.so` is located in the `odbc_cli_driver/linux/clidriver/lib` directory.

   In the DB2 server installation, `libdb2.so` is located in the `sqllib/lib` directory.
4. From the command prompt, test your setup by typing `python` to start the Python interpreter and entering code similar to that shown in the examples:
   - To test ibm_db:

     ```
     import ibm_db
     ibm_db_conn = ibm_db.connect('dsn=database', 'user', 'password')
     import ibm_db_dbi
     conn = ibm_db_dbi.Connection(ibm_db_conn)
     conn.tables('SYSCAT', '%')
     ```

   - To test ibm_db_sa:

     ```
     import sqlalchemy
     from sqlalchemy import *
     import ibm_db_sa.ibm_db_sa
     db2 = sqlalchemy.create_engine('ibm_db_sa://user:password@host.name.com:50000/database')
     metadata = MetaData()
     users = Table('users', metadata,
     Column('user_id', Integer, primary_key = True),
     Column('user_name', String(16), nullable = False),
     Column('email_address', String(60), key='email'),
     Column('password', String(20), nullable = False)
     )
     metadata.bind = db2
     metadata.create_all()
     users_table = Table('users', metadata, autoload=True, autoload_with=db2)
     users_table
     ```

   - To test ibm_db_django:
     a. Create a new Django project:

        ```
        django-admin.py startproject myproj
        ```

     b. Edit the settings.py file to configure access to DB2. Use any editor available on the system. An example on nix is:

        ```
        $ cd myproj
        $ vi settings.py

        DATABASE_ENGINE    = 'ibm_db_django'
        DATABASE_NAME      = 'mydb'
        DATABASE_USER      = 'db2inst1'
        DATABASE_PASSWORD  = 'ibmdb2'
        DATABASE_HOST      = 'localhost'
        DATABASE_PORT      = '50000'
        ```

     c. Add the listed lines in the tuple INSTALLED_APPS section of the settings.py file.

        ```
        'django.contrib.flatpages',
        'django.contrib.redirects',
        'django.contrib.comments',
        'django.contrib.admin',
        ```

     d. Run a test suite to confirm the configuration is correct:

```
python manage.py test
```

Results

The Python packages are now installed on your system and ready to use.

What to do next

Connect to the data server, and begin issuing SQL statements.

Application development in Python with ibm_db

The ibm_db API provides a variety of useful Python functions for accessing and manipulating data in an IBM data server database, including functions for connecting to a database, preparing and issuing SQL statements, fetching rows from result sets, calling stored procedures, committing and rolling back transactions, handling errors, and retrieving metadata.

Connecting to an IBM data server database in Python

Before you can execute SQL statements to create, update, delete, or retrieve data, you must connect to a database. You can use the ibm_db API to connect to a database through either a cataloged or uncataloged connection. To improve performance, you can also create a persistent connection.

Before you begin

- Set up the Python environment.
- Issue the `import ibm_db` command from your Python script.

Procedure

To return a connection resource that you can use to call SQL statements, call one of the listed functions:

Table 10. ibm_db connection functions

Function	Description
ibm_db.connect	Creates a nonpersistent connection.
ibm_db.pconnect	Creates a persistent connection. A persistent connection remains open after the initial Python script request, which allows subsequent Python requests to reuse the connection if they have an identical set of credentials.

The database value that you pass as an argument to these functions can be either a cataloged database name or a complete database connection string for a direct TCP/IP connection. You can specify optional arguments that control the timing of committing transactions, the case of the column names that are returned, and the cursor type.
If the connection attempt fails, you can retrieve diagnostic information by calling the ibm_db.conn_error or ibm_db.conn_errormsg function.
For more information about the ibm_db API, see http://code.google.com/p/ibm-db/wiki/APIs.

Example

Example 1: Connect to a local or cataloged database

Approach 1:

```
import ibm_db
conn = ibm_db.connect("dsn=name","username","password")
```

Approach 2:

```
import ibm_db
conn = ibm_db.connect("name","username","password")
```

Example 2: Connect to a cataloged or uncataloged database

```
import ibm_db
ibm_db.connect("DATABASE=name;HOSTNAME=host;PORT=60000;PROTOCOL=TCPIP;UID=username;
                PWD=password;", "", "")
```

What to do next

If the connection attempt is successful, you can use the connection resource when you call ibm_db functions that execute SQL statements. Next, you prepare and execute SQL statements.

Executing SQL statements in Python

After connecting to a database, use functions available in the ibm_db API to prepare and execute SQL statements. The SQL statements can contain static text, XQuery expressions, or parameter markers that represent variable input.

Preparing and executing a single SQL statement in Python:

To prepare and execute a single SQL statement, use the ibm_db.exec_immediate function. To avoid the security threat of SQL injection attacks, use the ibm_db.exec_immediate function only to execute SQL statements composed of static strings. Interpolation of Python variables representing user input into the SQL statement can expose your application to SQL injection attacks.

Before you begin

Obtain a connection resource by calling one of the connection functions in the ibm_db API. Refer to "Connecting to an IBM data server database in Python" on page 47.

Procedure

To prepare and execute a single SQL statement, call the ibm_db.exec_immediate function, passing the listed arguments:

connection
A valid database connection resource returned from the ibm_db.connect or ibm_db.pconnect function.

statement
A string that contains the SQL statement. This string can include an XQuery expression that is called by the XMLQUERY function.

options
Optional: A dictionary that specifies the type of cursor to return for result sets.

You can use this parameter to request a scrollable cursor for database servers that support this type of cursor. By default, a forward-only cursor is returned.

If the function call fails (returns `False`), you can use the ibm_db.stmt_error or ibm_db.stmt_errormsg function to retrieve diagnostic information about the error. If the function call succeeds, you can use the ibm_db.num_rows function to return the number of rows that the SQL statement returned or affected. If the SQL statement returns a result set, you can begin fetching the rows.
For more information about the ibm_db API, see http://code.google.com/p/ibm-db/wiki/APIs.

Example

Example 1: Execute a single SQL statement

```
import ibm_db
conn = ibm_db.connect("dsn=name","username","password")
stmt = ibm_db.exec_immediate(conn, "UPDATE employee SET bonus = '1000' WHERE job = 'MANAGER'")
print "Number of affected rows: ", ibm_db.num_rows(stmt)
```

Example 2: Execute an XQuery expression

```
import ibm_db
conn = ibm_db.connect("dsn=name","username","password")
if conn:
    sql = "SELECT XMLSERIALIZE(XMLQUERY('for $i in $t/address where $i/city = \"Olathe\" return <zip>
                    {$i/zip/text()}</zip>' passing c.xmlcol as \"t\") AS CLOB(32k)) FROM xml_test c WHERE id = 1"
    stmt = ibm_db.exec_immediate(conn, sql)
    result = ibm_db.fetch_both(stmt)
    while( result ):
        print "Result from XMLSerialize and XMLQuery:", result[0]
        result = ibm_db.fetch_both(stmt)
```

What to do next

If the SQL statement returns one or more result sets, you can begin fetching rows from the statement resource.

Preparing and executing SQL statements with variable input in Python:

To prepare and execute an SQL statement that includes variable input, use the ibm_db.prepare, ibm_db.bind_param, and ibm_db.execute functions. Preparing a statement improves performance because the database server creates an optimized access plan for data retrieval that it can reuse if the statement is executed again.

Before you begin

Obtain a connection resource by calling one of the connection functions in the ibm_db API. Refer to “Connecting to an IBM data server database in Python” on page 47.

Procedure

To prepare and execute an SQL statement that includes parameter markers:

1. Call the ibm_db.prepare function, passing the listed arguments:

   connection
   A valid database connection resource returned from the ibm_db.connect or ibm_db.pconnect function.

   statement
   A string that contains the SQL statement, including question marks (?) as parameter markers for column or predicate values that require variable input. This string can include an XQuery expression that is called by the XMLQUERY function.

options
Optional: A dictionary that specifies the type of cursor to return for result sets. You can use this parameter to request a scrollable cursor for database servers that support this type of cursor. By default, a forward-only cursor is returned.

If the function call succeeds, it returns a statement handle resource that you can use in subsequent function calls that are related to the query.

If the function call fails (returns `False`), you can use the ibm_db.stmt_error or ibm_db.stmt_errormsg function to retrieve diagnostic information about the error.

2. Optional: For each parameter marker in the SQL string, call the ibm_db.bind_param function, passing the listed arguments. Binding input values to parameter markers ensures that each input value is treated as a single parameter, which prevents SQL injection attacks.

   stmt
   The prepared statement returned by the call to the ibm_db.prepare function.

   parameter-number
   An integer that represents the position of the parameter marker in the SQL statement.

   variable
   The value to use in place of the parameter marker.

3. Call the ibm_db.execute function, passing the listed arguments:

   stmt
   A prepared statement returned from ibm_db.prepare.

   parameters
   A tuple of input parameters that match parameter markers contained in the prepared statement.

   For more information about the ibm_db API, see http://code.google.com/p/ibm-db/wiki/APIs.

Example

Prepare and execute a statement that includes variable input.

```
import ibm_db
conn = ibm_db.connect("dsn=name","username","password")
sql = "SELECT EMPNO, LASTNAME FROM EMPLOYEE WHERE EMPNO > ? AND EMPNO < ?"
stmt = ibm_db.prepare(conn, sql)
max = 50
min = 0
# Explicitly bind parameters
ibm_db.bind_param(stmt, 1, min)
ibm_db.bind_param(stmt, 2, max)
ibm_db.execute(stmt)
# Process results

# Invoke prepared statement again using dynamically bound parameters
param = max, min,
ibm_db.execute(stmt, param)
```

What to do next

If the SQL statement returns one or more result sets, you can begin fetching rows from the statement resource.

Fetching rows or columns from result sets in Python

After executing a statement that returns one or more result sets, use one of the functions available in the ibm_db API to iterate through the returned rows. If your result set includes columns that contain extremely large data (such as BLOB or CLOB data), you can retrieve the data on a column-by-column basis to avoid using too much memory.

Before you begin

You must have a statement resource returned by either the ibm_db.exec_immediate or ibm_db.execute function that has one or more associated result sets.

Procedure

To fetch data from a result set:

1. Fetch data from a result set by calling one of the fetch functions.

Table 11. ibm_db fetch functions

Function	Description
ibm_db.fetch_tuple	Returns a tuple, indexed by column position, representing a row in a result set. The columns are 0-indexed.
ibm_db.fetch_assoc	Returns a dictionary, indexed by column name, representing a row in a result set.
ibm_db.fetch_both	Returns a dictionary, indexed by both column name and position, representing a row in a result set.
ibm_db.fetch_row	Sets the result set pointer to the next row or requested row. Use this function to iterate through a result set.

   These functions accept the listed arguments:

   stmt
   A valid statement resource.

   row_number
   The number of the row that you want to retrieve from the result set. Specify a value for this parameter if you requested a scrollable cursor when you called the ibm_db.exec_immediate or ibm_db.prepare function. With the default forward-only cursor, each call to a fetch method returns the next row in the result set.

2. Optional: If you called the ibm_db.fetch_row function, for each iteration through the result set, retrieve a value from a specified column by calling the ibm_db.result function. You can specify the column by passing either an integer that represents the position of the column in the row (starting with 0) or a string that represents the name of the column.
3. Continue fetching rows until the fetch method returns False, which indicates that you have reached the end of the result set.

   For more information about the ibm_db API, see http://code.google.com/p/ibm-db/wiki/APIs.

Example

Example 1: Fetch rows from a result set by calling the ibm_db.fetch_both function

```
import ibm_db

conn = ibm_db.connect( "dsn=name", "username", "password" )
sql = "SELECT * FROM EMPLOYEE"
stmt = ibm_db.exec_immediate(conn, sql)
dictionary = ibm_db.fetch_both(stmt)
while dictionary != False:
    print "The ID is : ",  dictionary["EMPNO"]
    print "The Name is : ", dictionary[1]
    dictionary = ibm_db.fetch_both(stmt)
```

Example 2: Fetch rows from a result set by calling the ibm_db.fetch_tuple function

```
import ibm_db

conn = ibm_db.connect( "dsn=name", "username", "password" )
sql = "SELECT * FROM EMPLOYEE"
stmt = ibm_db.exec_immediate(conn, sql)
tuple = ibm_db.fetch_tuple(stmt)
while tuple != False:
    print "The ID is : ", tuple[0]
    print "The name is : ", tuple[1]
    tuple = ibm_db.fetch_tuple(stmt)
```

Example 3: Fetch rows from a result set by calling the ibm_db.fetch_assoc function

```
import ibm_db

conn = ibm_db.connect( "dsn=name", "username", "password" )
sql = "SELECT * FROM EMPLOYEE"
stmt = ibm_db.exec_immediate(conn, sql)
dictionary = ibm_db.fetch_assoc(stmt)
while dictionary != False:
    print "The ID is : ", dictionary["EMPNO"]
    print "The name is : ", dictionary["FIRSTNME"]
    dictionary = ibm_db.fetch_assoc(stmt)
```

Example 4: Fetch columns from a result set

```
import ibm_db

conn = ibm_db.connect( "dsn=name", "username", "password" )
sql = "SELECT * FROM EMPLOYEE
stmt = ibm_db.exec_immediate(conn, sql)
while ibm_db.fetch_row(stmt) != False:
    print "The Employee number is : ",  ibm_db.result(stmt, 0)
    print "The Name is : ", ibm_db.result(stmt, "NAME")
```

What to do next

When you are ready to close the connection to the database, call the ibm_db.close function. If you attempt to close a persistent connection that you created with ibm_db.pconnect, the close request returns True, and the connection remains available for the next caller.

Calling stored procedures in Python

To call a stored procedure from a Python application, you prepare and execute an SQL CALL statement. The procedure that you call can include input parameters (IN), output parameters (OUT), and input and output parameters (INOUT).

Before you begin

Obtain a connection resource by calling one of the connection functions in the ibm_db API. Refer to "Connecting to an IBM data server database in Python" on page 47.

Procedure

To call a stored procedure:

1. Call the ibm_db.prepare function, passing the listed arguments:

   connection
   A valid database connection resource returned from ibm_db.connect or ibm_db.pconnect.

   statement
   A string that contains the SQL CALL statement, including parameter markers (?) for any input or output parameters.

   options
   Optional: A dictionary that specifies the type of cursor to return for result sets. You can use this parameter to request a scrollable cursor for database servers that support this type of cursor. By default, a forward-only cursor is returned.

2. For each parameter marker in the CALL statement, call the ibm_db.bind_param function, passing the listed arguments:

   stmt
   The prepared statement returned by the call to the ibm_db.prepare function.

   parameter-number
   An integer that represents the position of the parameter marker in the SQL statement.

   variable
   The name of the Python variable that will hold the output.

   parameter-type
   A constant that specifies whether to bind the Python variable to the SQL parameter as an input parameter (SQL_PARAM_INPUT), an output parameter (SQL_PARAM_OUTPUT), or a parameter that accepts input and returns output (SQL_PARAM_INPUT_OUTPUT).

   This step binds each parameter marker to the name of a Python variable that will hold the output.

3. Call the ibm_db.execute function, passing the prepared statement as an argument.

   For more information about the ibm_db API, see http://code.google.com/p/ibm-db/wiki/APIs.

Example

Prepare and execute an SQL CALL statement.

```
import ibm_db

conn = ibm_db.connect("dsn=sample","username","password")
if conn:
      sql = 'CALL match_animal(?, ?, ?)'
      stmt = ibm_db.prepare(conn, sql)
```

```
name = "Peaches"
second_name = "Rickety Ride"
weight = 0
ibm_db.bind_param(stmt, 1, name, ibm_db.SQL_PARAM_INPUT)
ibm_db.bind_param(stmt, 2, second_name, ibm_db.SQL_PARAM_INPUT_OUTPUT)
ibm_db.bind_param(stmt, 3, weight, ibm_db.SQL_PARAM_OUTPUT)

print "Values of bound parameters _before_ CALL:"
print "  1: %s 2: %s 3: %d\n" % (name, second_name, weight)

if ibm_db.execute(stmt):
  print "Values of bound parameters _after_ CALL:"
  print "  1: %s 2: %s 3: %d\n" % (name, second_name, weight)
```

What to do next

If the procedure call returns one or more result sets, you can begin fetching rows from the statement resource.

Retrieving multiple result sets from a stored procedure in Python

When a single call to a stored procedure returns more than one result set, you can use the ibm_db.next_result function of the ibm_db API to retrieve the result sets.

Before you begin

You must have a statement resource returned by the ibm_db.exec_immediate or ibm_db.execute function that has multiple result sets.

Procedure

To retrieve multiple result sets:

1. Fetch rows from the first result set returned from the procedure by calling one of the listed ibm_db fetch functions, passing the statement resource as an argument. (The first result set that is returned from the procedure is associated with the statement resource.)

Table 12. ibm_db fetch functions

Function	Description
ibm_db.fetch_tuple	Returns a tuple, indexed by column position, representing a row in a result set. The columns are 0-indexed.
ibm_db.fetch_assoc	Returns a dictionary, indexed by column name, representing a row in a result set.
ibm_db.fetch_both	Returns a dictionary, indexed by both column name and position, representing a row in a result set.
ibm_db.fetch_row	Sets the result set pointer to the next row or requested row. Use this function to iterate through a result set.

2. Retrieve the subsequent result sets by passing the original statement resource as the first argument to the ibm_db.next_result function. You can fetch rows from the statement resource until no more rows are available in the result set.

   The ibm_db.next_result function returns False when no more result sets are available or if the procedure did not return a result set.

For more information about the ibm_db API, see http://code.google.com/p/ibm-db/wiki/APIs.

Example

Retrieve multiple result sets from a stored procedure.

```
import ibm_db
conn = ibm_db.connect( "dsn=sample", "user", "password" )
if conn:
    sql = 'CALL sp_multi()'
    stmt = ibm_db.exec_immediate(conn, sql)
    row = ibm_db.fetch_assoc(stmt)
    while row != False :
        print "The value returned : ", row
        row = ibm_db.fetch_assoc(stmt)

    stmt1 = ibm_db.next_result(stmt)
    while stmt1 != False:
        row = ibm_db.fetch_assoc(stmt1)
        while row != False :
            print "The value returned : ", row
            row = ibm_db.fetch_assoc(stmt1)
        stmt1 = ibm_db.next_result(stmt)
```

What to do next

When you are ready to close the connection to the database, call the ibm_db.close function. If you attempt to close a persistent connection that you created by using ibm_db.pconnect, the close request returns True, and the IBM data server client connection remains available for the next caller.

Commit modes in Python applications

You can control how groups of SQL statements are committed by specifying a commit mode for a connection resource. The ibm_db API supports two commit modes: autocommit and manual commit.

autocommit mode

In autocommit mode, each SQL statement is a complete transaction, which is automatically committed. Autocommit mode helps prevent locking escalation issues that can impede the performance of highly scalable web applications. By default, the ibm_db API opens every connection in autocommit mode.

You can turn on autocommit mode after disabling it by calling `ibm_db.autocommit(`*conn*`, ibm_db.SQL_AUTOCOMMIT_ON)`, where *conn* is a valid connection resource.

Calling the ibm_db.autocommit function might affect the performance of your Python scripts because it requires additional communication between Python and the database management system.

manual commit mode

In manual commit mode, the transaction ends when you call the ibm_db.commit or ibm_db.rollback function. This means that all statements executed on the same connection between the start of a transaction and the call to the commit or rollback function are treated as a single transaction.

Manual commit mode is useful if you might have to roll back a transaction that contains one or more SQL statements. If you exectue SQL statements

in a transaction and the script ends without explicitly committing or rolling back the transaction, the ibm_db extension automatically rolls back any work performed in the transaction.

You can turn off autocommit mode when you create a database connection by using the { ibm_db.SQL_ATTR_AUTOCOMMIT: ibm_db.SQL_AUTOCOMMIT_OFF } setting in the ibm_db.connect or ibm_db.pconnect options array. You can also turn off autocommit mode for a connection resource by calling ibm_db.autocommit(*conn*, ibm_db.SQL_AUTOCOMMIT_OFF), where *conn* is a valid connection resource.

For more information about the ibm_db API, see http://code.google.com/p/ibm-db/wiki/APIs.

Example

Turn off autocomit mode and end the transaction when ibm_db.commit or ibm_db.rollback is called.

```
import ibm_db

array = { ibm_db.SQL_ATTR_AUTOCOMMIT : ibm_db.SQL_AUTOCOMMIT_OFF }
conn = ibm_db.pconnect("dsn=SAMPLE", "user", "password", array)
sql = "DELETE FROM EMPLOYEE"
try:
   stmt = ibm_db.exec_immediate(conn, sql)
except:
   print "Transaction couldn't be completed."
   ibm_db.rollback(conn)
else:
   ibm_db.commit(conn)
   print "Transaction complete."
```

Error-handling functions in Python

Sometimes errors happen when you attempt to connect to a database or issue an SQL statement. The username or password might be incorrect, a table or column name might be misspelled, or the SQL statement might be invalid. The ibm_db API provides error-handling functions to help you recover gracefully from these situations.

Connection errors

Use one of the listed functions to retrieve diagnostic information if a connection attempt fails.

Table 13. ibm_db functions for handling connection errors

Function	Description
ibm_db.conn_error	Retrieves the SQLSTATE returned by the last connection attempt
ibm_db. conn_errormsg	Retrieves a descriptive error message appropriate for an application error log

SQL errors

Use one of the listed functions to retrieve diagnostic information if an attempt to prepare or execute an SQL statement or to fetch a result from a result set fails.

Table 14. ibm_db functions for handling SQL errors

Function	Description
ibm_db.stmt_error	Retrieves the SQLSTATE returned by the last attempt to prepare or execute an SQL statement or to fetch a result from a result set
ibm_db.stmt_errormsg	Retrieves a descriptive error message appropriate for an application error log

For more information about the ibm_db API, see http://code.google.com/p/ibm-db/wiki/APIs.

Example

Example 1: Handle connection errors

```
import ibm_db
try:
    conn = ibm_db.connect("dsn=sample","user","password")
except:
    print "no connection:", ibm_db.conn_errormsg()
else:
    print "The connection was successful"
```

Example 2: Handle SQL errors

```
import ibm_db
conn = ibm_db.connect( "dsn=sample", "user", "password")
sql = "DELETE FROM EMPLOYEE"
try:
   stmt = ibm_db.exec_immediate(conn, sql)
except:
   print "Transaction couldn't be completed:" , ibm_db.stmt_errormsg()
else:
   print "Transaction complete."
```

Database metadata retrieval functions in Python

You can use functions in the ibm_db API to retrieve metadata for IBM databases.

Before calling these functions, you must set up the Python environment, issue `import_db` in your Python script, and obtain a connection resource by calling the ibm_db.connect or ibm_db.pconnect function.

Important: Calling metadata functions uses a significant amount of space. If possible, cache the results of your calls for use in subsequent calls.

Table 15. ibm_db metadata retrieval functions

Function	Description
ibm_db.client_info	Returns a read-only object with information about the IBM data server client
ibm_db.column_privileges	Returns a result set listing the columns and associated privileges for a table
ibm_db.columns	Returns a result set listing the columns and associated metadata for a table
ibm_db.foreign_keys	Returns a result set listing the foreign keys for a table

Table 15. ibm_db metadata retrieval functions (continued)

Function	Description
ibm_db.primary_keys	Returns a result set listing the primary keys for a table
ibm_db.procedure_columns	Returns a result set listing the parameters for one or more stored procedures
ibm_db.procedures	Returns a result set listing the stored procedures registered in a database
ibm_db.server_info	Returns a read-only object with information about the IBM data server
ibm_db.special_columns	Returns a result set listing the unique row identifier columns for a table
ibm_db.statistics	Returns a result set listing the index and statistics for a table
ibm_db.table_privileges	Returns a result set listing the tables in a database and the associated privileges

For more information about the ibm_db API, see http://code.google.com/p/ibm-db/wiki/APIs.

Example

Example 1: Display information about the IBM data server client

```
import ibm_db

conn = ibm_db.connect("dsn=sample", "user", "password")
client = ibm_db.client_info(conn)

if client:
    print "DRIVER_NAME: string(%d) \"%s\"" % (len(client.DRIVER_NAME), client.DRIVER_NAME)
    print "DRIVER_VER: string(%d) \"%s\"" % (len(client.DRIVER_VER), client.DRIVER_VER)
    print "DATA_SOURCE_NAME: string(%d) \"%s\"" % (len(client.DATA_SOURCE_NAME), client.DATA_SOURCE_NAME)
    print "DRIVER_ODBC_VER: string(%d) \"%s\"" % (len(client.DRIVER_ODBC_VER), client.DRIVER_ODBC_VER)
    print "ODBC_VER: string(%d) \"%s\"" % (len(client.ODBC_VER), client.ODBC_VER)
    print "ODBC_SQL_CONFORMANCE: string(%d) \"%s\"" % (len(client.ODBC_SQL_CONFORMANCE), client.ODBC_SQL_CONFORMANCE)
    print "APPL_CODEPAGE: int(%s)" % client.APPL_CODEPAGE
    print "CONN_CODEPAGE: int(%s)" % client.CONN_CODEPAGE
    ibm_db.close(conn)
else:
    print "Error."
```

Example 2: Display information about the IBM data server

```
import ibm_db

conn = ibm_db.connect("dsn=sample", "user", "password")
server = ibm_db.server_info(conn)

if server:
    print "DBMS_NAME: string(%d) \"%s\"" % (len(server.DBMS_NAME), server.DBMS_NAME)
    print "DBMS_VER: string(%d) \"%s\"" % (len(server.DBMS_VER), server.DBMS_VER)
    print "DB_NAME: string(%d) \"%s\"" % (len(server.DB_NAME), server.DB_NAME)
    ibm_db.close(conn)
else:
    print "Error."
```

Chapter 4. Developing Ruby on Rails applications

The IBM_DB Ruby driver and Rails adapter

With the introduction of support for the Ruby on Rails framework, Rails applications can now access data on IBM data servers.

Collectively known as the IBM_DB gem, the IBM_DB Ruby driver and Rails adapter allows Ruby applications to access the listed database management systems:

- DB2 for Linux, UNIX, and Windows Version 9 and later
- DB2 Universal Database (DB2 UDB) Version 8 for Linux, UNIX, and Windows
- DB2 UDB Version 5, Release 1 (and later) for AS/400® and iSeries®, through DB2 Connect
- DB2 for z/OS, Version 8 and Version 9, through DB2 Connect
- Informix Dynamic Server, Version 11.10 and later

  Note: Client applications must use IBM Data Server Driver Version 9.5 or later when accessing Informix Dynamic Server Version 11.10. Previous versions are not supported. Client applications can also use IBM Data Server Runtime Client or IBM Data Server Client.

The IBM_DB Ruby driver can be used to connect to and access data from the IBM data servers mentioned previously. The IBM_DB Ruby adapter allows any database-backed Rails application to interface with IBM data servers.

For more information about IBM Ruby projects and the RubyForge open source community, see http://rubyforge.org/projects/rubyibm/

For a list of installation requirements for DB2 database products, see http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.qb.server.doc/doc/r0025127.html

For a list of installation requirements for IBM Informix Dynamic Server, see http://publib.boulder.ibm.com/infocenter/idshelp/v111/topic/com.ibm.expr.doc/ids_in_004x.html

For information about downloading an IBM Data Server Driver Package (DS Driver), see http://www.ibm.com/software/data/support/data-server-clients/index.html.

Getting started with IBM data servers on Rails

To start developing Ruby on Rails applications with IBM data servers, you must set up the Rails environment with IBM data servers. To get started, you can download the free version of DB2 and start developing Rails applications using DB2.

Before you begin

To ensure that numeric values in quotations are handled correctly, you must use Version 9.1 Fix Pack 2 (or later) of one of the listed client types: IBM Data Server

Driver Package, IBM Data Server Client, or IBM Data Server Driver for ODBC and CLI.

Procedure

To set up your environment and get started with IBM_DB:

1. Download and install DB2 or IBM Informix from http://www.ibm.com/software/data/servers/.
2. Download and install the latest version of Ruby from http://www.ruby-lang.org/en/downloads/.
3. Install the Rails gem and its dependencies by issuing the gem install command:

```
gem install rails —-include-dependencies
```

What to do next

You are now ready to install the IBM_DB Ruby driver and Rails adapter as a gem. If you want, you can also set up an integrated development environment (IDE) for Rails.

Setting up an integrated development environment for Rails

Rails requires no special file formats or integrated development environments (IDEs); you can get started with a command line prompt and a text editor. However, various IDEs are now available with Rails support, such as RadRails, which is a Rails environment for Eclipse.

About this task

For more information about RadRails, see http://www.radrails.org/.

Procedure

To set up an Eclipse based IDE for Ruby on Rails (RoR) development:

1. Install Eclipse from http://www.eclipse.org/downloads/.
2. Install the Eclipse plug-ins from the Eclipse remote update sites:
   a. Ruby Development Tools from http://rubyeclipse.sourceforge.net/download.rdt.html
   b. RubyRails IDE feature from http://radrails.sourceforge.net/update
   c. Subclipse plug-in from http://subclipse.tigris.org/update

Installing the IBM_DB adapter and driver as a Ruby gem

Ruby Gems is the standard packaging and installation framework for libraries and applications in the Ruby runtime environment. A single file for each bundle is called a gem, which complies to the package format. This package is then distributed and stored in a central repository, allowing simultaneous deployment of multiple versions of the same library or application.

About this task

If you are connecting from Ruby to a DB2 database server on a remote machine, then you need one of the listed DB2 clients on the machine where you are installing, running or executing Ruby: IBM Data Server Driver Package or IBM Data Server Runtime Client or IBM Data Server Client.

If you are connecting from Ruby/Ruby on Rails to a DB2 database server on the local machine, you don't need to install a DB2 database client.

Similar to package management and bundles (.rpm, .deb) used in Linux distributions, these gems can also be queried, installed, uninstalled, and manipulated through the gem end-user utility.

The gem utility can seamlessly query the remote RubyForge central repository and look up and install any of the many readily available utilities. When the IBM_DB gem is installed, this functionality is immediately accessible from any script (or application) in the Ruby runtime environment, through:

```
require 'ibm_db'
```

or on Windows:

```
require 'mswin32/ibm_db'
```

Procedure

To install the IBM_DB adapter and driver as a Ruby gem:

1. On Linux, UNIX, and Mac OS X platforms, set environment variables and optionally source the DB2 profile:
   a. Issue the `export` commands to set the environment variables **IBM_DB_INCLUDE** and **IBM_DB_LIB**:

      ```
      $ export IBM_DB_INCLUDE=DB2HOME/include
      $ export IBM_DB_LIB=DB2HOME/lib
      ```

      where *DB2HOME* is the directory where the IBM data server is installed. For example:

      ```
      $ export IBM_DB_INCLUDE=/home/db2inst1/sqllib/include
      $ export IBM_DB_LIB=/home/db2inst1/sqllib/lib
      ```

      If you are using ibm_db 1.0.0 or earlier, instead of setting IBM_DB_INCLUDE, you must set the environment variable **IBM_DB_DIR** to *DB2HOME*.

      More about setting environment variables:

      Depending on the architecture for which the IBM data server is installed, the `lib` directory under *DB2HOME* is a link to either `lib32` or `lib64`. You can set **IBM_DB_LIB** according to the architecture for which Ruby is compiled. For a 32-bit architecture, set **IBM_DB_LIB** to the `lib32` directory under *DB2HOME*. For a 64-bit architecture set **IBM_DB_LIB** to the `lib64` directory under *DB2HOME*.

      Note:

      In IBM Data Server Driver Package, *DB2HOME* refers to the directory where the client package is untarred, for example, the `odbc_cli_driver/linux/clidriver` directory.

      In the DB2 server installation, *DB2HOME* refers to the `sqllib` directory under your DB2 instance.
2. On all supported platforms, issue the gem install command to install the IBM_DB adapter and driver:

   ```
   $ gem install ibm_db
   ```

3. Before running any ruby script that connects to DB2, you need to ensure that the IBM DB2 Ruby driver can access the CLI driver libdb2.so, which is part of your DB2 Server setup or DB2 client setup. Failing to do so will result in a `missing libraries - libdb2.so.1` error when you run your Ruby program. You do this by adding the folder where the libdb2.so file resides to the **LD_LIBRARY_PATH** environmental variable on Linux or AIX® platforms for the userid under which Ruby will be run.

   When using IBM Data Server Driver Package, the `libdb2.so` file is in the `odbc_cli_driver/linux/clidriver/lib` directory.

   In the DB2 server installation, libdb2.so is in the `sqllib/lib/` directory.

   Note: For a 32-bit architecture, set **LD_LIBRARY_PATH** to the `lib32` directory under *DB2HOME*. For a 64-bit architecture set **LD_LIBRARY_PATH** to the `lib64` directory under *DB2HOME*.

Results

The IBM_DB gem is now installed on your workstation.

Verifying installation of the IBM_DB gem with DB2 Express-C

To verify installation of the IBM_DB gem with DB2 Express-C, you connect to the database, issue a SELECT statement, and then fetch the first row of the result set.

Procedure

Use the `gem` commands to install and verify the installation of the IBM_DB gem with Ruby-1.8.6 patch level 111 on a Windows or Linux operating system. The output of the commands is also shown.

- To perform the installation, issue the command **gem install ibm_db**. For example:

```
D:\>gem install ibm_db
Select which gem to install for your platform (i386-mswin32)
1. ibm_db 1.0.1 (ruby)
2. ibm_db 1.0.1 (mswin32)
2. ibm_db 1.0.0 (ruby)
3. ibm_db 1.0.0 (mswin32)
4. Skip this gem
5. Cancel installation
> 2
Successfully installed ibm_db-1.0.0-mswin32
Installing ri documentation for ibm_db-1.0.0-mswin32...
Installing RDoc documentation for ibm_db-1.0.0-mswin32...
```

  Note: The examples in this topic include version information to demonstrate the installation. However, when you run the installation, you can choose from the two latest versions of the gem that are available.
  The IBM_DB gem is now installed on your machine.

- To verify the installation, run the listed commands.

  You can follow this process to verify installation against IBM Informix, DB2 Database for Linux, UNIX, and Windows, IBM DB2 for IBM i, and DB2 for z/OS. You can use DB2 Connect to access IBM DB2 for IBM i and DB2 for z/OS data servers.

```
C:\>irb
irb(main):001:0> require 'mswin32/ibm_db' (if using Linux based
platform then issue require 'ibm_db')
=>true
irb(main):002:0> conn = IBM_DB::connect
```

```
'devdb','username','password' (Here 'devdb' is the database cataloged in
client's database directory)
=> #<IBM_DB::Connection:0x2dddf40>
irb(main):003:0> stmt = IBM_DB::exec conn,'select * from cars'
=> #<IBM_DB::Statement:0x2beaabc>
irb(main):004:0> IBM_DB::fetch_assoc stmt (will fetch the first row of
the result set)
```

What to do next

If these commands run successfully, the gem is installed correctly, and you can begin building Rails applications.

Verifying installation with IBM data servers on Rails applications

To verify that the IBM_DB driver and adapter are installed correctly, you test IBM_DB driver access by connecting to an IBM data server and issuing a SELECT statement, and then you test IBM_DB adapter access by building and running a sample Rails application.

Procedure

To verify installation:

1. Install the latest version of the IBM_DB gem.
2. Test IBM_DB driver access.

   For example, to test the access to an i5 data server through the IBM_DB driver (and underlying DB2 Connect and IBM Data Server Driver for ODBC and CLI):

   ```
   D:\ws\RoR\TeamRoom>irb
   irb(main):001:0> require 'mswin32/ibm_db'
   => true
   irb(main):002:0> conn = IBM_DB::connect 'testdb', 'user', 'pass'
   => #<IBM_DB::Connection:0x2f79d40>
   irb(main):003:0> stmt = IBM_DB::exec conn, 'select * from qsys2.qsqptabl'
   => #<IBM_DB::Statement:0x2f762f8>
   irb(main):004:0> IBM_DB::fetch_assoc stmt
   ```

3. Test IBM_DB adapter access.

   To test access to an IBM data server through the IBM_DB adapter, perform the listed steps to build a sample Rails application.

   a. Create a new Rails application by issuing the listed command:

      ```
      C:\>rails newapp --database=ibm_db
      create
      create app/controllers
      create app/helpers
      create app/models
      create app/views/layouts
      create config/environments
      create config/initializers
      create db
      [......]
      create log/server.log
      create log/production.log
      create log/development.log
      create log/test.log
      ```

   b. Change to the newly created directory, `newapp`:

      ```
      C:\>cd newapp
      ```

   c. Configure connections for the Rails application by editing the `database.yml` file. For more information, see "Configuring Rails application connections to IBM data servers" on page 64.

If you are using a version before Rails 2.0, you must register the IBM_DB adapter to the list of connection adapters in the Rails framework by manually adding `ibm_db` to the list of connection adapters in `<RubyHome>\gems\1.8\gems\activerecord-1.15.6\lib\active_record.rb` at approximately line 77:

```
RAILS_CONNECTION_ADAPTERS = %w( mysql postgresql sqlite firebird
sqlserver db2 oracle sybase openbase frontbase ibm_db )
```

d. Create a model and scaffold by issuing the `ruby` command:

```
C:\>ruby script/generate scaffold Tool name:string model_num:integer
exists app/models/
exists app/controllers/
[....]
create db/migrate
create db/migrate/20080716103959_create_tools.rb
```

e. Issue the Rails `migrate` command to create the table (tools) in the database (devdb):

```
C:\ >rake db:migrate
(in C:/ruby trials/newapp)
== 20080716111617 CreateTools: migrating
=====================================
-- create_table(:tools)
-> 0.5320s
== 20080716111617 CreateTools: migrated (0.5320s)
```

The Rails application can now access the Tools table and perform operations on it.

f. Issue the `ruby` command to test the application:

```
C:\ruby trials\newapp>ruby script/console
Loading development environment (Rails 2.1.0)
>> tool = Tool.new
=> #<Tool id: nil, name: nil, model_num: nil, created_at: nil,
updated_at: nil>
>> tool.name = 'chistel'
=> "chistel"
>> tool.model_num = '007'
=> "007"
>> tool.save
=> true
>> Tool.find :all
=> [#<Tool id: 100, name: "chistel", model_num: 7, created_at:
"2008-07-16 11:29:35", updated_at: "2008-07-16 11:29:35">]
>>
```

Configuring Rails application connections to IBM data servers

You configure database connections for a Rails application by specifying connection details in the `database.yml` file.

Procedure

To configure host data server connections for a Rails application:

Edit the database configuration details in *rails_application_path*`\config\database.yml`, and specify the listed connection attributes:

```
# The IBM_DB Adapter requires the native Ruby driver (ibm_db)
    # for IBM data servers (ibm_db.so).
    # +config+ the hash passed as an initializer argument content:
    # == mandatory parameters
    #   adapter:     'ibm_db'        // IBM_DB Adapter name
    #   username:    'db2user'       // data server (database) user
    #   password:    'secret'        // data server (database) password
    #   database:    'DEVDB'         // remote database name (or catalog entry alias)
```

```
    # == optional (highly recommended for data server auditing and monitoring purposes)
    #   schema:      'rails123'      // name space qualifier
    #   account:     'tester'        // OS account (client workstation)
    #   app_user:    'test11'        // authenticated application user
    #   application: 'rtests'        // application name
    #   workstation: 'plato'         // client workstation name
    # == remote TCP/IP connection (required when no local database catalog entry available)
    #   host:        'Socrates'      // fully qualified hostname or IP address
    #   port:        '50000'         // data server TCP/IP port number
    #
    # When schema is not specified, the username value is used instead.
```

Note: Changes to connection information in this file are applied when the Rails environment is initialized during server startup. Any changes that you make after initialization do not affect the connections that are created.

Schema, account, app_user, application and workstation are not supported for IBM Informix.

IBM Ruby driver and trusted contexts

The IBM_DB Ruby driver supports trusted contexts by using connection string keywords.

Trusted contexts provide a way of building much faster and more secure three-tier applications. The user's identity is always preserved for auditing and security purposes. When you require secure connections, trusted contexts improve performance because you do not have to get new connections.

Example

The example establishes a trusted connection and switches the user on the same connection.

```
def trusted_connection(database,hostname,port,auth_user,auth_pass,tc_user,tc_pass)
  dsn = "DATABASE=#{database};HOSTNAME=#{hostname};PORT=#{port};PROTOCOL=TCPIP;UID=#{auth_user};PWD=#{auth_pass};"
  conn_options = {IBM_DB::SQL_ATTR_USE_TRUSTED_CONTEXT => IBM_DB::SQL_TRUE}
  tc_options = {IBM_DB::SQL_ATTR_TRUSTED_CONTEXT_USERID => tc_user, IBM_DB::SQL_ATTR_TRUSTED_CONTEXT_PASSWORD => tc_pass}
  tc_conn = IBM_DB.connect dsn, '', '', conn_options
  if tc_conn
    puts "Trusted connection established successfully."
    val = IBM_DB.get_option tc_conn, IBM_DB::SQL_ATTR_USE_TRUSTED_CONTEXT, 1
    if val
      userBefore = IBM_DB.get_option tc_conn, IBM_DB::SQL_ATTR_TRUSTED_CONTEXT_USERID, 1
      #do some work as user 1
      #....
      #....
      #switch the user
      result = IBM_DB.set_option tc_conn, tc_options, 1
      userAfter = IBM_DB.get_option tc_conn, IBM_DB::SQL_ATTR_TRUSTED_CONTEXT_USERID, 1
      if userBefore != userAfter
        puts "User has been switched."
        #do some work as user 2
        #....
        #....
      end
    end
    IBM_DB.close tc_conn
  else
    puts "Attempt to connect failed due to: #{IBM_DB.conn_errormsg}"
  end
end
```

IBM_DB Rails adapter dependencies and consequences

The IBM_DB adapter (ibm_db_adapter.rb) has a direct dependency on the IBM_DB driver, which uses IBM Data Server Driver for ODBC and CLI to connect to IBM data servers. The IBM Call Level Interface (CLI) is a callable SQL interface to IBM data servers, which is Open Database Connectivity (ODBC) compliant.

This dependency has several ramifications for the IBM_DB adapter and driver.

- Installation of IBM Data Server Driver for ODBC and CLI, which meets the IBM_DB requirement, is required.

IBM Data Server Driver for ODBC and CLI is included with a full DB2 database install, or you can obtain it separately

Note: The IBM Data Server Driver for ODBC and CLI is included in the listed client packages:

- IBM Data Server Client
- IBM Data Server Runtime Client
- IBM Data Server Driver Package

- Driver behavior can be modified outside of a Rails application by using CLI keywords.

  Certain transactional behavior can be altered outside the Rails application by using these CLI keywords. For example, CLI keywords can be used to set the current schema or alter transactional elements such as turning off autocommit behavior. For more information about CLI keywords, see the listed links:

  For Version 9: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.apdv.cli.doc/doc/r0007964.htm

  For Version 9.5: http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/topic/com.ibm.db2.luw.apdv.cli.doc/doc/r0007964.html

  For Version 9.7: http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.cli.doc/doc/r0007964.html

- Any diagnostic gathering requires CLI driver tracing.

  Because all requests through the IBM_DB driver are implemented through IBM Data Server Driver for ODBC and CLI, the CLI trace facility can identify problems for applications that use the IBM_DB adapter and driver.

  A CLI trace captures all of the API calls made by an application to the IBM Data Server Driver for ODBC and CLI (including all input parameters), and it captures all of the values returned from the driver to the application. It is an interface trace that captures how an application interacts with the IBM Data Server Driver for ODBC and CLI and offers information about the inner workings of the driver.

The IBM_DB Ruby driver and Rails adapter are not supported on JRuby

The IBM_DB adapter is not supported on JRuby.

The IBM_DB adapter is not supported on JRuby because (as stated in the JRuby Wiki, "Getting Started"): "Many Gems will work fine in JRuby, however some Gems build native C libraries as part of their install process. These Gems will not work in JRuby unless the Gem has also provided a Java equivalent to the native library." For more information, see http://kenai.com/projects/jruby/pages/GettingStarted.

The IBM_DB adapter relies on the IBM_DB Ruby driver (C extension) and the IBM Data Server Driver for ODBC and CLI to access databases on IBM data servers. Alternatively, you can either use the regular C implementation of Ruby, or use JDBC_adapter to access databases.

ActiveRecord-JDBC versus IBM_DB adapter

Update 0.6.0 and later of the IBM_DB gem provides a slightly different handling of the required numeric values quoting.

While the previous version of the adapter was attempting to screen out such usage of quotation marks on numeric values to conform to DB2 data server expectations on different platforms, the new implementation replaces the workaround with a permanent fix in the IBM data server client. This not only enables IBM data servers across platforms but provides a more reliable handling of all Rails APIs that could escape previous screening. The workaround provided by the previous version of the adapter is by its nature quite brittle, due to fluid developments in the Rails framework components (ActiveRecord). It is also known that certain Rails APIs managed to escape the screening of those overridden methods, so the workaround used in the ActiveRecord-JDBC adapter might require handling of some additional cases.

The JRuby runtime does not benefit from the same fix due to its inner specific interaction with the data servers. IBM DB2 for IBM i does not exhibit this issue (fixed in V5R3 and V5R4) and the same is true regarding IBM Informix. For the time being, until JRuby and ActiveRecord-JDBC adapter matures, the best alternative is to use the "classic Ruby" (C implementation) and the IBM_DB adapter/driver. A fix in the ActiveRecord-JDBC adapter could also be considered, which could emulate the previous handling that the IBM_DB adapter was providing.

Heap size considerations with DB2 on Rails

Rails applications on DB2 require the **applheapsz** database configuration parameter to be set to values above 1024.

You must set this parameter for each database for which you will be running DB2 on Rails applications. Use the `db2 update db cfg` command to update the **applheapsz** parameter:

```
db2 update db cfg for database_name using APPLHEAPSZ 1024
```

To activate this parameter, you must restart your DB2 instance.

Appendix A. Overview of the DB2 technical information

DB2 technical information is available in multiple formats that can be accessed in multiple ways.

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

- DB2 Information Center
  - Topics (Task, concept and reference topics)
  - 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 hardcopy 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/e-business/linkweb/publications/servlet/pbi.wss. English and translated DB2 Version 10.1 manuals in PDF format 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.

The form number increases each time a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed below.

Note: The *DB2 Information Center* is updated more frequently than either the PDF or the hard-copy books.

Table 16. DB2 technical information

Name	Form Number	Available in print	Last updated
Administrative API Reference	SC27-3864-00	Yes	April, 2012
Administrative Routines and Views	SC27-3865-00	No	April, 2012
Call Level Interface Guide and Reference Volume 1	SC27-3866-00	Yes	April, 2012
Call Level Interface Guide and Reference Volume 2	SC27-3867-00	Yes	April, 2012
Command Reference	SC27-3868-00	Yes	April, 2012
Database Administration Concepts and Configuration Reference	SC27-3871-00	Yes	April, 2012
Data Movement Utilities Guide and Reference	SC27-3869-00	Yes	April, 2012
Database Monitoring Guide and Reference	SC27-3887-00	Yes	April, 2012
Data Recovery and High Availability Guide and Reference	SC27-3870-00	Yes	April, 2012
Database Security Guide	SC27-3872-00	Yes	April, 2012
DB2 Workload Management Guide and Reference	SC27-3891-00	Yes	April, 2012
Developing ADO.NET and OLE DB Applications	SC27-3873-00	Yes	April, 2012
Developing Embedded SQL Applications	SC27-3874-00	Yes	April, 2012
Developing Java Applications	SC27-3875-00	Yes	April, 2012
Developing Perl, PHP, Python, and Ruby on Rails Applications	SC27-3876-00	No	April, 2012
Developing User-defined Routines (SQL and External)	SC27-3877-00	Yes	April, 2012
Getting Started with Database Application Development	GI13-2046-00	Yes	April, 2012

Table 16. DB2 technical information (continued)

Name	Form Number	Available in print	Last updated
Getting Started with DB2 Installation and Administration on Linux and Windows	GI13-2047-00	Yes	April, 2012
Globalization Guide	SC27-3878-00	Yes	April, 2012
Installing DB2 Servers	GC27-3884-00	Yes	April, 2012
Installing IBM Data Server Clients	GC27-3883-00	No	April, 2012
Message Reference Volume 1	SC27-3879-00	No	April, 2012
Message Reference Volume 2	SC27-3880-00	No	April, 2012
Net Search Extender Administration and User's Guide	SC27-3895-00	No	April, 2012
Partitioning and Clustering Guide	SC27-3882-00	Yes	April, 2012
pureXML Guide	SC27-3892-00	Yes	April, 2012
Spatial Extender User's Guide and Reference	SC27-3894-00	No	April, 2012
SQL Procedural Languages: Application Enablement and Support	SC27-3896-00	Yes	April, 2012
SQL Reference Volume 1	SC27-3885-00	Yes	April, 2012
SQL Reference Volume 2	SC27-3886-00	Yes	April, 2012
Text Search Guide	SC27-3888-00	Yes	April, 2012
Troubleshooting and Tuning Database Performance	SC27-3889-00	Yes	April, 2012
Upgrading to DB2 Version 10.1	SC27-3881-00	Yes	April, 2012
What's New for DB2 Version 10.1	SC27-3890-00	Yes	April, 2012
XQuery Reference	SC27-3893-00	No	April, 2012

Table 17. DB2 Connect-specific technical information

Name	Form Number	Available in print	Last updated
DB2 Connect Installing and Configuring DB2 Connect Personal Edition	SC27-3861-00	Yes	April, 2012
DB2 Connect Installing and Configuring DB2 Connect Servers	SC27-3862-00	Yes	April, 2012
DB2 Connect User's Guide	SC27-3863-00	Yes	April, 2012

Displaying SQL state help from the command line processor

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

Procedure

To start 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

Documentation for other versions of DB2 products is found in separate information centers on ibm.com®.

About this task

For DB2 Version 10.1 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v10r1.

For DB2 Version 9.8 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/.

For DB2 Version 9.7 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/.

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

For DB2 Version 9.1 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9/.

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

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

A locally installed DB2 Information Center must be updated periodically.

Before you begin

A DB2 Version 10.1 Information Center must already be installed. For details, see the “Installing the DB2 Information Center using the DB2 Setup wizard” topic in *Installing DB2 Servers*. All prerequisites and restrictions that applied to installing the Information Center also apply to updating the Information Center.

About this task

An existing DB2 Information Center can be updated automatically or manually:

- Automatic updates update existing Information Center features and languages. One benefit of automatic updates is that the Information Center is unavailable for a shorter time compared to during a manual update. In addition, automatic updates can be set to run as part of other batch jobs that run periodically.
- Manual updates can be used to update existing Information Center features and languages. Automatic updates reduce the downtime during the update process, however you must use the manual process when you want to add features or languages. For example, a local Information Center was originally installed with both English and French languages, and now you want to also install the German language; a manual update will install German, as well as, update the existing Information Center features and languages. However, a manual update requires you to manually stop, update, and restart the Information Center. The Information Center is unavailable during the entire update process. In the automatic update process the Information Center incurs an outage to restart the Information Center after the update only.

This topic details the process for automatic updates. For manual update instructions, see the "Manually updating the DB2 Information Center installed on your computer or intranet server" topic.

Procedure

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

1. On Linux operating systems,
   a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `/opt/ibm/db2ic/V10.1` directory.
   b. Navigate from the installation directory to the `doc/bin` directory.
   c. Run the `update-ic` script:
      ```
      update-ic
      ```
2. On Windows operating systems,
   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 10.1` 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 `update-ic.bat` file:
      ```
      update-ic.bat
      ```

Results

The DB2 Information Center restarts automatically. If updates were available, the Information Center displays the new and updated topics. If Information Center updates were not available, a message is added to the log. The log file is located in `doc\eclipse\configuration` directory. The log file name is a randomly generated number. For example, `1239053440785.log`.

Manually updating the DB2 Information Center installed on your computer or intranet server

If you have installed the DB2 Information Center locally, you can obtain and install documentation updates from IBM.

About this task

Updating your locally installed *DB2 Information Center* manually 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 apply updates. The Workstation version of the DB2 Information Center always runs in stand-alone mode. .
2. Use the Update feature to see what updates are available. If there are updates that you must install, you can use the Update feature to obtain and install them

   Note: If your environment requires installing the *DB2 Information Center* updates on a machine that is not connected to the internet, mirror the update site to a local file system by 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 get 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 2008, Windows Vista (and higher), the commands listed later in this section must be run as an administrator. To open a command prompt or graphical tool with full administrator privileges, right-click the shortcut and then select **Run as administrator**.

Procedure

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 **DB2 Information Center** service and select **Stop**.
   - On Linux, enter the following command:
     ```
     /etc/init.d/db2icdv10 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 10.1 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/V10.1` 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 opens to display the stand-alone Information Center.

3. Click the **Update** button (). (JavaScript must be enabled in your browser.) On the right panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the installation process, check that the selections you want to install, then click **Install Updates**.
5. After the installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center:
   - On Windows, navigate to the `doc\bin` directory within the installation directory, and run the `help_end.bat` file:

     ```
     help_end.bat
     ```

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

     ```
     help_end
     ```

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

     ```
     /etc/init.d/db2icdv10 start
     ```

Results

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

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 database 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/db2luw/v10r1/.

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 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.

DB2 troubleshooting information

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

DB2 documentation
Troubleshooting information can be found in the *Troubleshooting and Tuning Database Performance* or the Database fundamentals section of the *DB2 Information Center*, which contains:

- Information about how to isolate and identify problems with DB2 diagnostic tools and utilities.
- Solutions to some of the most common problem.
- Advice to help solve other problems you might encounter with your DB2 database products.

IBM Support Portal
See the IBM Support Portal 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 IBM Support Portal at http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows

Terms and conditions

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

Applicability: These terms and conditions are in addition to any terms of use for the IBM website.

Personal use: You may reproduce these publications for your personal, noncommercial 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.

Rights: 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.

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

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. Information about non-IBM products is based on information available at the time of first publication of this document and is subject to change.

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

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

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

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

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 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, changes, or both in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to websites not owned by IBM are provided for convenience only and do not in any manner serve as an endorsement of those

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

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

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
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
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 contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

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

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

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel, Intel logo, Intel Inside, Intel Inside logo, Celeron, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

Index

A

ActiveRecord-JDBC adapter
  IBM_DB adapter comparison 67
application design
  prototyping in Perl 1
autocommit function (ibm_db) 55

B

bind_param function (ibm_db)
  calling 49, 53

C

CALL statement
  PHP 23, 38
  Python 53
client_info function (ibm_db) 57
close function (ibm_db)
  fetching from result sets 51
  retrieving multiple result sets 54
column_privileges function (ibm_db) 57
columns function (ibm_db) 57
commit function (ibm_db) 55
commit modes
  PHP applications 25, 40
  Python applications 55
conn_error function (ibm_db) 56
conn_errormsg function (ibm_db) 56
connect function (ibm_db) 47
connect method (Perl DBI) 2
connections
  Rails applications 64

D

DB2 Information Center
  updating 72, 74
  versions 72
db2_autocommit function (ibm_db2) 25
db2_bind_param function (ibm_db2)
  calling stored procedures 23
  executing SQL statements with variable input 17
  inserting large objects 19
  preparing SQL statements with variable input 17
db2_client_info function (ibm_db2) 27
db2_close function (ibm_db2) 20
db2_column_privileges function (ibm_db2) 27
db2_columns function (ibm_db2) 27
db2_commit function (ibm_db2) 25
db2_conn_error function (ibm_db2) 26
db2_conn_errormsg function (ibm_db2) 26
db2_connect function (ibm_db2) 14
db2_exec function (ibm_db2) 16
db2_execute function (ibm_db2)
  calling stored procedures 23
  executing SQL statements 17
  inserting large objects 19
db2_fetch_array function (ibm_db2)
  fetching data from a result set 20
  retrieving multiple result sets 24
db2_fetch_assoc function (ibm_db2)
  fetching data from a result set 20
  retrieving multiple result sets 24
db2_fetch_both function (ibm_db2)
  fetching data from a result set 20
  retrieving multiple result sets 24
db2_fetch_object function (ibm_db2) 22
  fetching data from a result set 20
db2_fetch_row function (ibm_db2)
  fetching data from a result set 20
  retrieving multiple result sets 24
db2_foreign_keys function (ibm_db2) 27
db2_next_result function (ibm_db2)
  retrieving multiple result sets 24
db2_pconnect function (ibm_db2) 14
db2_prepare function (ibm_db2)
  calling stored procedures 23
  inserting large objects 19
  preparing SQL statements 17
db2_primary_keys function (ibm_db2) 27
db2_procedure_columns function (ibm_db2) 27
db2_procedures function (ibm_db2) 27
db2_result function (ibm_db2) 20
db2_rollback function (ibm_db2) 25
db2_server_info function (ibm_db2) 27
db2_special_columns function (ibm_db2) 27
db2_statistics function (ibm_db2) 27
db2_stmt_error function (ibm_d2b) 26
db2_stmt_errormsg function (ibm_db2) 26
db2_table_privileges function (ibm_db2) 27
DB2::DB2 driver
  downloads 1
  pureXML support 5
  resources 1
disconnect method (Perl DBI) 2
Django
  IBM data server environment setup 44
documentation
  overview 69
  PDF files 69
  printed 69
  terms and conditions of use 76
dynamic SQL
  Perl support 1

E

err method 4
errors
  Perl 4
  PHP 26, 41
  Python 56
errstr method 4
exec_immediate function (ibm_db) 48
execute function (ibm_db)
  calling stored procedures 53
  executing SQL statements with variable input 49
execute method (Perl DBI) 3

F

fetch_assoc function (ibm_db)
  fetching columns 51
  fetching multiple result sets 54
  fetching rows 51
fetch_both function (ibm_db)
  fetching columns 51
  fetching multiple result sets 54
  fetching rows 51
fetch_row function (ibm_db)
  fetching columns 51
  fetching multiple result sets 54
  fetching rows 51
fetch_tuple function (ibm_db)
  fetching columns 51
  fetching multiple result sets 54
  fetching rows 51
fetchrow method (Perl DBI) 3
foreign_keys function (ibm_db) 57
functions
  PHP
    db2_autocommit 25
    db2_bind_param 17, 19, 23
    db2_client_info 27
    db2_close 20, 24
    db2_column_privileges 27
    db2_columns 27
    db2_commit 25
    db2_conn_error 26
    db2_conn_errormsg 26
    db2_connect 14
    db2_exec 16
    db2_execute 17, 19, 23
    db2_fetch_array 20, 24
    db2_fetch_assoc 20, 24
    db2_fetch_both 20, 24
    db2_fetch_object 20, 22
    db2_fetch_row 20, 24
    db2_foreign_keys 27
    db2_next_result 24
    db2_pconnect 14
    db2_prepare 17, 19, 23
    db2_primary_keys 27
    db2_procedure_columns 27
    db2_procedures 27
    db2_result 20
    db2_rollback 25
    db2_server_info 27
    db2_special_columns 27
    db2_statistics 27
    db2_stmt_error 26
    db2_stmt_errormsg 26
    db2_table_privileges 27
  Python
    ibm_db.autocommit 55
    ibm_db.bind_param 49, 53
    ibm_db.client_info 57
    ibm_db.close 51, 54
    ibm_db.column_privileges 57
    ibm_db.columns 57
    ibm_db.commit 55
    ibm_db.conn_error 56
    ibm_db.conn_errormsg 56
    ibm_db.connect 47
    ibm_db.exec_immediate 48
    ibm_db.execute 49, 53
    ibm_db.fetch_assoc 51, 54
functions *(continued)*
  Python *(continued)*
    ibm_db.fetch_both 51, 54
    ibm_db.fetch_row 51, 54
    ibm_db.fetch_tuple 51, 54
    ibm_db.foreign_keys 57
    ibm_db.next_result 54
    ibm_db.pconnect 47
    ibm_db.prepare 49, 53
    ibm_db.primary_keys 57
    ibm_db.procedure_columns 57
    ibm_db.procedures 57
    ibm_db.result 51
    ibm_db.rollback 55
    ibm_db.server_info 57
    ibm_db.special_columns 57
    ibm_db.statistics 57
    ibm_db.stmt_error 56
    ibm_db.stmt_errormsg 56
    ibm_db.table_privileges 57

H

help
  SQL statements 72
host variables
  Perl 3

I

ibm_db API
  details 43
  overview 47
IBM_DB Ruby driver and Rails adapter
  ActiveRecord-JDBC adapter comparison 67
  dependencies 65
  details 59
  environment setup 59
  installation verification
    DB2 Express-C 62
    IBM data servers 63
  integrated development environment setup 60
  JRuby support 66
  Ruby gem installation 60
  trusted contexts 65
ibm_db_dbi API
  details 43
ibm_db_sa adaptor
  details 43
ibm_db2 API
  details 9
  PHP application development 14
  trusted contexts 15

J

JRuby
  IBM_DB Ruby driver and Rails adapter 66

L

large objects (LOBs)
  fetching
    PHP 22, 37

large objects (LOBs) *(continued)*
inserting
PHP 19, 34

M

metadata
retrieval
PHP 27
Python 57
methods
Perl
connect 2
disconnect 2
err 4
errstr 4
execute 3
fetchrow 3
prepare 3
state 4
PHP
PDO::beginTransaction 40
PDO::commit 40
PDO::exec 32
PDO::prepare 33, 34, 38
PDO::query 32
PDO::rollBack 40
PDOStatement::bindColumn 37
PDOStatement::bindParam 33, 34, 38
PDOStatement::execute 33, 34, 38
PDOStatement::fetch 35, 37, 39
PDOStatement::fetchAll 35, 39
PDOStatement::fetchColumn 35
PDOStatement::nextRowset 39

N

next_result function (ibm_db) 54
notices 79

P

parameter markers
Perl 4
pconnect function (ibm_db) 47
pdo_ibm
details 9
developing PHP applications 30
PDO::beginTransaction method (PDO) 40
PDO::commit method (PDO) 40
PDO::exec method (PDO) 32
PDO::prepare method (PDO) 33, 34, 38
PDO::query method (PDO) 32
PDO::rollBack method (PDO) 40
PDOStatement::bindColumn method (PDO) 37
PDOStatement::bindParam method (PDO) 33, 34, 38
PDOStatement::execute method (PDO) 33, 34, 38
PDOStatement::fetch method (PDO) 35, 37, 39
PDOStatement::fetchAll method (PDO) 35, 39
PDOStatement::fetchColumn method (PDO) 35
PDOStatement::nextRowset method (PDO) 39
Perl
connecting to a database 2
documentation 1
downloads 1
drivers 1
Perl *(continued)*
errors 4
fetching rows 3
methods
connect 2
disconnect 2
err 4
errstr 4
execute 3
fetchrow 3
prepare 3
state 4
overview 1
parameter markers 4
problem reporting 1
pureXML support 5
restrictions 5
sample programs 7, 8
SQLCODEs 4
SQLSTATEs 4
PHP
application development 9, 14
connecting to database 14, 31
database metadata retrieval 27
developing applications with PDO 30
documentation 10
downloads 10
error handling 26, 41
extensions for IBM data servers 9
fetching large objects 22, 37
fetching rows 20, 35
functions
db2_autocommit 25
db2_bind_param 23
db2_client_info 27
db2_close 20, 24
db2_column_privileges 27
db2_columns 27
db2_commit 25
db2_conn_error 26
db2_conn_errormsg 26
db2_connect 14
db2_exec 16
db2_execute 23
db2_fetch_array 20, 24
db2_fetch_assoc 20, 24
db2_fetch_both 20, 24
db2_fetch_object 20, 22
db2_fetch_row 20, 24
db2_foreign_keys 27
db2_next_result 24
db2_pconnect 14
db2_prepare 23
db2_primary_keys 27
db2_procedure_columns 27
db2_procedures 27
db2_result 20
db2_rollback 25
db2_server_info 27
db2_special_columns 27
db2_statistics 27
db2_stmt_error 26
db2_stmt_errormsg 26
db2_table_privileges 27
IBM data server environment setup (Windows) 10
ibm_db2 API
connecting to a database 14

PHP *(continued)*
ibm_db2 API *(continued)*
overview 14
large objects 19, 34
methods
PDO::beginTransaction 40
PDO::commit 40
PDO::exec 32
PDO::prepare 33, 34, 38
PDO::query 32
PDO::rollBack 40
PDOStatement::bindColumn 37
PDOStatement::bindParam 33, 34, 38
PDOStatement::execute 33, 34, 38
PDOStatement::fetch 35, 37, 39
PDOStatement::fetchAll 35, 39
PDOStatement::fetchColumn 35
PDOStatement::nextRowset 39
PDO_IBM extension
connecting to database 31
issuing SQL statements 32
procedures 23, 38
setup
Linux 12
overview 10
UNIX 12
SQL statements 16, 17, 19, 20, 32, 33, 34, 35, 37
stored procedures
calling 23, 38
retrieving results 24, 39
transactions 25, 40
trusted contexts
overview 15
prepare function (ibm_db) 49, 53
prepare method (Perl DBI) 3
primary_keys function (ibm_db) 57
problem determination
information available 76
tutorials 76
procedure_columns function (ibm_db) 57
procedures
PHP 23, 38
Python 53
procedures function (ibm_db) 57
pureXML
DB2::DB2 driver 5
Python
API documentation 44
application development 43, 47
connecting to database 47
database metadata retrieval 57
downloading extensions 44
error handling 56
extensions for IBM data servers 43
fetching rows 51
functions
ibm_db.autocommit 55
ibm_db.bind_param 49, 53
ibm_db.client_info 57
ibm_db.close 51, 54
ibm_db.column_privileges 57
ibm_db.columns 57
ibm_db.commit 55
ibm_db.conn_error 56
ibm_db.conn_errormsg 56
ibm_db.connect 47
ibm_db.exec_immediate 48
Python *(continued)*
functions *(continued)*
ibm_db.execute 49, 53
ibm_db.fetch_assoc 51, 54
ibm_db.fetch_both 51, 54
ibm_db.fetch_row 51, 54
ibm_db.fetch_tuple 51, 54
ibm_db.foreign_keys 57
ibm_db.next_result 54
ibm_db.pconnect 47
ibm_db.prepare 49, 53
ibm_db.primary_keys 57
ibm_db.procedure_columns 57
ibm_db.procedures 57
ibm_db.result 51
ibm_db.rollback 55
ibm_db.server_info 57
ibm_db.special_columns 57
ibm_db.statistics 57
ibm_db.stmt_error 56
ibm_db.stmt_errormsg 56
ibm_db.table_privileges 57
IBM data server environment setup 44
ibm_db 47
procedures 53
SQL statements 48, 49
stored procedures
calling 53
retrieving results 54
transactions 55

R

RadRails
IBM data server on Rails setup 60
Rails adapter
dependencies 65
details 59
getting started 59
IBM_DB adapter and driver installation 60
installation verification
DB2 Express-C 62
IBM data servers 63
integrated development environment setup 60
JRuby support 66
Rails applications
connection configuration 64
result function (ibm_db) 51
rollback function (ibm_db) 55
rows
fetching
Perl 3
PHP 20, 35
Python 51
Ruby driver
details 59
getting started 59
IBM_DB adapter and driver installation 60
installation verification
DB2 Express-C 62
IBM data servers 63
integrated development environment setup 60
JRuby support 66
trusted contexts 65
Ruby on Rails
heap size issues 67

S

samples
  Perl 7, 8
server_info function (ibm_db) 57
special_columns function (ibm_db) 57
SQL statements
  help
    displaying 72
  PHP 16, 17, 19, 20, 32, 33, 34, 35, 37
  Python 48, 49
SQLAlchemy
  adapter for IBM data servers 43
  downloading extension 44
  IBM data server environment setup 44
state method 4
static SQL
  unsupported in Perl 5
statistics function (ibm_db) 57
stmt_error function (ibm_db) 56
stmt_errormsg function (ibm_db) 56
stored procedures
  PHP
    calling 23, 38
    retrieving results 24, 39
  Python
    calling 53
    retrieving results 54

T

table_privileges function (ibm_db) 57
terms and conditions
  publications 76
transactions
  PHP 25, 40
  Python 55
troubleshooting
  online information 76
  tutorials 76
trusted contexts
  IBM_DB Ruby driver support
    details 65
  PHP applications 15
tutorials
  list 75
  problem determination 76
  pureXML 75
  troubleshooting 76

U

updates
  DB2 Information Center 72, 74

Printed in USA

SC27-3876-00

Spine information:

IBM DB2 10.1 for Linux, UNIX, and Windows

Developing Perl, PHP, Python, and Ruby on Rails Applications