IBM DB2, Cloudscape and Apache Derby

Introduction

These functions enable you to access IBM DB2 Universal Database, IBM Cloudscape, and Apache Derby databases using the DB2 Call Level Interface (DB2 CLI).

Installing/Configuring

Requirements

To connect to IBM DB2 Universal Database for Linux, UNIX, and Windows, or IBM Cloudscape, or Apache Derby, you must install an IBM DB2 Universal Database client on the same computer on which you are running PHP. The extension has been developed and tested with DB2 Version 8.2.

To connect to IBM DB2 Universal Database for z/OS or iSeries, you also require IBM DB2 Connect or the equivalent DRDA gateway software.

Requirements on Linux or Unix

The user invoking the PHP executable or SAPI must specify the DB2 instance before accessing these functions. You can set the name of the DB2 instance in *php.ini* using the *ibm_db2.instance_name* configuration option, or you can source the DB2 instance profile before invoking the PHP executable.

If you created a DB2 instance named *db2inst1* in */home/db2inst1/*, for example, you can add the following line to *php.ini*:
```
ibm_db2.instance_name=db2inst1
```
If you do not set this option in *php.ini*, you must issue the following command to modify your environment variables to enable access to DB2:
```
bash$ source /home/db2inst1/sqllib/db2profile
```
To enable your PHP-enabled Web server to access these functions, you must either set the *ibm_db2.instance_name* configuration option in *php.ini*, or source the DB2 instance environment in your Web server start script (typically */etc/init.d/httpd* or */etc/init.d/apache*).

Installation

To build the ibm_db2 extension, the DB2 application development header files and libraries must be installed on your system. DB2 does not install these by default, so you may have to return to your DB2 installer and add this option. The header files are included with the DB2 Application Development Client freely available for download from the IBM DB2 Universal Database » support site.

If you add the DB2 application development header files and libraries to a Linux or Unix operating system on which DB2 was already installed, you must issue the command *db2iupdt -e* to update the symbolic links to the header files and libraries in your DB2 instances.

ibm_db2 is a » PECL extension, so follow the instructions in Installation of PECL extensions to install the ibm_db2 extension for PHP. Issue the *configure* command to point to the location of your DB2 header files and libraries as follows:
```
bash$ ./configure --with-IBM_DB2=/path/to/DB2
```
The *configure* command defaults to */opt/IBM/db2/V8.1*.

Note
Note for IIS users If you are using the ibm_db2 driver with Microsoft Internet Information Server (IIS) you may have to do the following: • Install DB2 with extended operating system security. • Add the PHP binary path to the PATH system environment variable (default C:\php\). • Create another system environment variable equal to the path where the PHP.INI file is located (eg: PHPRC = C:\php\). • Add the IUSR_COMPUTERNAME to the DB2USERS group.

Runtime Configuration

The behaviour of these functions is affected by settings in *php.ini*.

ibm_db2 Configure Options

Name	Default	Changeable	Changelog
ibm_db2.binmode	"1"	PHP_INI_ALL	
ibm_db2.i5_allow_commit	"0"	PHP_INI_SYSTEM	Available since ibm_db2 1.4.9.
ibm_db2.i5_dbcs_alloc	"0"	PHP_INI_SYSTEM	Available since ibm_db2 1.5.0.
ibm_db2.instance_name	NULL	PHP_INI_SYSTEM	Available since ibm_db2 1.0.2.
ibm_db2.i5_all_pconnect	"0"	PHP_INI_SYSTEM	Available since ibm_db2 1.6.5.

Here's a short explanation of the configuration directives.

ibm_db2.binmode integer

This option controls the mode used for converting to and from binary data in the PHP application.

- 1 (DB2_BINARY)
- 2 (DB2_CONVERT)
- 3 (DB2_PASSTHRU)

ibm_db2.i5_allow_commit integer
This option controls the commit mode used for i5 schema collections in the PHP application.

- 0 no commit (see i5_commit for override)
- 1 allow commit (see i5_commit for override)

ibm_db2.i5_dbcs_alloc integer
This option controls the internal ibm_db2 allocation scheme for large DBCS column buffers.

- 0 no expanded allocations (see i5_dbcs_alloc for override)
- 1 use expanded allocations (see i5_dbcs_alloc for override)

ibm_db2.instance_name string
On Linux and UNIX operating systems, this option defines the name of the instance to use for cataloged database connections. If this option is set, its value overrides the DB2INSTANCE environment variable setting. This option is ignored on Windows operating systems.

ibm_db2.i5_all_pconnect integer
This option overrides i5 db2_connect full open and close in the PHP application. When ibm_db2.i5_all_pconnect = 1, all db2 connections become persistent (db2_pconnect). On i5/OS, db2_pconnect performs dramatically better with lower machine stress over db2_connect. This is a convenience override of db2_connect to evoke db2_pconnect without PHP source code changes.

- 0 db2_connect default full open and close
- 1 db2_connect override to db2_pconnect for persistent connection only

Resource Types

The ibm_db2 extension returns connection resources, statement resources, and result set resources.

Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

DB2_BINARY (integer)
Specifies that binary data shall be returned as is. This is the default mode.

DB2_CONVERT (integer)
Specifies that binary data shall be converted to a hexadecimal encoding and returned as an ASCII string.

DB2_PASSTHRU (integer)
Specifies that binary data shall be converted to a **NULL** value.

DB2_SCROLLABLE (integer)
Specifies a scrollable cursor for a statement resource. This mode enables random access to rows in a result set, but currently is supported only by IBM DB2 Universal Database.

DB2_FORWARD_ONLY (integer)
Specifies a forward-only cursor for a statement resource. This is the default cursor type and is supported on all database servers.

DB2_PARAM_IN (integer)
Specifies the PHP variable should be bound as an IN parameter for a stored procedure.

DB2_PARAM_OUT (integer)
Specifies the PHP variable should be bound as an OUT parameter for a stored procedure.

DB2_PARAM_INOUT (integer)
Specifies the PHP variable should be bound as an INOUT parameter for a stored procedure.

DB2_PARAM_FILE (integer)
Specifies that the column should be bound directly to a file for input.

DB2_AUTOCOMMIT_ON (integer)
Specifies that autocommit should be turned on.

DB2_AUTOCOMMIT_OFF (integer)
Specifies that autocommit should be turned off.

DB2_DOUBLE (integer)
Specifies that the variable should be bound as a DOUBLE, FLOAT, or REAL data type.

DB2_LONG (integer)
Specifies that the variable should be bound as a SMALLINT, INTEGER, or BIGINT

data type.

DB2_CHAR (integer)
Specifies that the variable should be bound as a CHAR or VARCHAR data type.

DB2_CASE_NATURAL (integer)
Specifies that column names will be returned in their natural case.

DB2_CASE_LOWER (integer)
Specifies that column names will be returned in lower case.

DB2_CASE_UPPER (integer)
Specifies that column names will be returned in upper case.

DB2_DEFERRED_PREPARE_ON (integer)
Specifies that deferred prepare should be turned on for the specified statement resource.

DB2_DEFERRED_PREPARE_OFF (integer)
Specifies that deferred prepare should be turned off for the specified statement resource.

IBM DB2 Functions

db2_autocommit

db2_autocommit -- Returns or sets the AUTOCOMMIT state for a database connection

Description

mixed **db2_autocommit** (resource `$connection` [, bool `$value`])

Sets or gets the AUTOCOMMIT behavior of the specified connection resource.

Parameters

`connection`
A valid database connection resource variable as returned from db2_connect() or db2_pconnect().

`value`
One of the following constants:
DB2_AUTOCOMMIT_OFF
Turns AUTOCOMMIT off.

DB2_AUTOCOMMIT_ON
Turns AUTOCOMMIT on.

Return Values

When db2_autocommit() receives only the *`connection`* parameter, it returns the current state of AUTOCOMMIT for the requested connection as an integer value. A value of 0 indicates that AUTOCOMMIT is off, while a value of 1 indicates that AUTOCOMMIT is on.

When db2_autocommit() receives both the *`connection`* parameter and *`autocommit`* parameter, it attempts to set the AUTOCOMMIT state of the requested connection to the corresponding state. Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #1 - Retrieving the AUTOCOMMIT value for a connection

In the following example, a connection which has been created with AUTOCOMMIT turned off is tested with the db2_autocommit() function.

```
<?php
$options = array('autocommit' => DB2_AUTOCOMMIT_OFF);
$conn = db2_connect($database, $user, $password, $options);
```

```
$ac = db2_autocommit($conn);
if ($ac == 0) {
   print "$ac -- AUTOCOMMIT is off.";
} else {
   print "$ac -- AUTOCOMMIT is on.";
}
?>
```

The above example will output:

```
0 -- AUTOCOMMIT is off.
```

Example #2 - Setting the AUTOCOMMIT value for a connection

In the following example, a connection which was initially created with AUTOCOMMIT turned off has its behavior changed to turn AUTOCOMMIT on.

```
<?php
$options = array('autocommit' => DB2_AUTOCOMMIT_OFF);
$conn = db2_connect($database, $user, $password, $options);

// Turn AUTOCOMMIT on
$rc = db2_autocommit($conn, DB2_AUTOCOMMIT_ON);
if ($rc) {
   print "Turning AUTOCOMMIT on succeeded.\n";
}

// Check AUTOCOMMIT state
$ac = db2_autocommit($conn);
if ($ac == 0) {
   print "$ac -- AUTOCOMMIT is off.";
} else {
   print "$ac -- AUTOCOMMIT is on.";
}
?>
```

The above example will output:

```
Turning AUTOCOMMIT on succeeded.
1 -- AUTOCOMMIT is on.
```

See Also

- db2_connect()
- db2_pconnect()

db2_bind_param

db2_bind_param -- Binds a PHP variable to an SQL statement parameter

Description

bool **db2_bind_param** (resource `$stmt`, int `$parameter-number`, string `$variable-name` [, int `$parameter-type` [, int `$data-type` [, int `$precision` [, int `$scale`]]]])

Binds a PHP variable to an SQL statement parameter in a statement resource returned by db2_prepare(). This function gives you more control over the parameter type, data type, precision, and scale for the parameter than simply passing the variable as part of the optional input array to db2_execute().

Parameters

`stmt`
A prepared statement returned from db2_prepare().

`parameter-number`
Specifies the 1-indexed position of the parameter in the prepared statement.

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

`parameter-type`
A constant specifying whether the PHP variable should be bound to the SQL parameter as an input parameter (*DB2_PARAM_IN*), an output parameter (*DB2_PARAM_OUT*), or as a parameter that accepts input and returns output (*DB2_PARAM_INOUT*). To avoid memory overhead, you can also specify *DB2_PARAM_FILE* to bind the PHP variable to the name of a file that contains large object (BLOB, CLOB, or DBCLOB) data.

`data-type`
A constant specifying the SQL data type that the PHP variable should be bound as: one of *DB2_BINARY*, *DB2_CHAR*, *DB2_DOUBLE*, or *DB2_LONG*.

`precision`
Specifies the precision with which the variable should be bound to the database. This parameter can also be used for retrieving XML output values from stored procedures. A non-negative value specifies the maximum size of the XML data that will be retrieved from the database. If this parameter is not used, a default of 1MB will be assumed for retrieving the XML output value from the stored procedure.

`scale`
Specifies the scale with which the variable should be bound to the database.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #3 - Binding PHP variables to a prepared statement

The SQL statement in the following example uses two input parameters in the WHERE clause. We call db2_bind_param() to bind two PHP variables to the corresponding SQL parameters. Notice that the PHP variables do not have to be declared or assigned before the call to db2_bind_param(); in the example, *$lower_limit* is assigned a value before the call to db2_bind_param(), but *$upper_limit* is assigned a value after the call to db2_bind_param(). The variables must be bound and, for parameters that accept input, must have any value assigned, before calling db2_execute().

```
<?php

$sql = 'SELECT name, breed, weight FROM animals
   WHERE weight > ? AND weight < ?';
$conn = db2_connect($database, $user, $password);
$stmt = db2_prepare($conn, $sql);

// We can declare the variable before calling db2_bind_param()
$lower_limit = 1;

db2_bind_param($stmt, 1, "lower_limit", DB2_PARAM_IN);
db2_bind_param($stmt, 2, "upper_limit", DB2_PARAM_IN);

// We can also declare the variable after calling db2_bind_param()
$upper_limit = 15.0;

if (db2_execute($stmt)) {
   while ($row = db2_fetch_array($stmt)) {
       print "{$row[0]}, {$row[1]}, {$row[2]}\n";
   }
}
?>
```

The above example will output:

```
Pook, cat, 3.2
Rickety Ride, goat, 9.7
Peaches, dog, 12.3
```

Example #4 - Calling stored procedures with IN and OUT parameters

The stored procedure match_animal in the following example accepts three different parameters:

- an input (IN) parameter that accepts the name of the first animal as input

- an input-output (INOUT) parameter that accepts the name of the second animal as input and returns the string *TRUE* if an animal in the database matches that name
- an output (OUT) parameter that returns the sum of the weight of the two identified animals

In addition, the stored procedure returns a result set consisting of the animals listed in alphabetic order starting at the animal corresponding to the input value of the first parameter and ending at the animal corresponding to the input value of the second parameter.

```
<?php

$sql = 'CALL match_animal(?, ?, ?)';
$conn = db2_connect($database, $user, $password);
$stmt = db2_prepare($conn, $sql);

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

db2_bind_param($stmt, 1, "name", DB2_PARAM_IN);
db2_bind_param($stmt, 2, "second_name", DB2_PARAM_INOUT);
db2_bind_param($stmt, 3, "weight", DB2_PARAM_OUT);

print "Values of bound parameters _before_ CALL:\n";
print "  1: {$name} 2: {$second_name} 3: {$weight}\n\n";

if (db2_execute($stmt)) {
   print "Values of bound parameters _after_ CALL:\n";
   print "  1: {$name} 2: {$second_name} 3: {$weight}\n\n";

   print "Results:\n";
   while ($row = db2_fetch_array($stmt)) {
       print "  {$row[0]}, {$row[1]}, {$row[2]}\n";
   }
}
?>
```

The above example will output:

```
Values of bound parameters _before_ CALL:
 1: Peaches 2: Rickety Ride 3: 0

Values of bound parameters _after_ CALL:
 1: Peaches 2: TRUE 3: 22

Results:
 Peaches, dog, 12.3
 Pook, cat, 3.2
 Rickety Ride, goat, 9.7
```

Example #5 - Inserting a binary large object (BLOB) directly from a file

The data for large objects are typically stored in files, such as XML documents or audio files. Rather than reading an entire file into a PHP variable, and then binding that PHP

variable into an SQL statement, you can avoid some memory overhead by binding the file directly to the input parameter of your SQL statement. The following example demonstrates how to bind a file directly into a BLOB column.

```
<?php
$stmt = db2_prepare($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);
?>
```

See Also

- db2_execute()
- db2_prepare()

db2_client_info

db2_client_info -- Returns an object with properties that describe the DB2 database client

Description

object **db2_client_info** (resource `$connection`)

This function returns an object with read-only properties that return information about the DB2 database client. The following table lists the DB2 client properties:

DB2 client properties

Property name	Return type	Description
APPL_CODEPAGE	int	The application code page.
CONN_CODEPAGE	int	The code page for the current connection.
DATA_SOURCE_NAME	string	The data source name (DSN) used to create the current connection to the database.
DRIVER_NAME	string	The name of the library that implements the DB2 Call Level Interface (CLI) specification.
DRIVER_ODBC_VER	string	The version of ODBC that the DB2 client supports. This returns a string "MM.mm" where *MM* is the major version and *mm* is the minor version. The DB2 client always returns "03.51".
DRIVER_VER	string	The version of the client, in the form of a string "MM.mm.uuuu" where *MM* is the major version, *mm* is the minor version, and *uuuu* is the update. For example, "08.02.0001" represents major version 8, minor version 2, update 1.
ODBC_SQL_CONFORMAN	string	

CE		The level of ODBC SQL grammar supported by the client: MINIMUM Supports the minimum ODBC SQL grammar. CORE Supports the core ODBC SQL grammar. EXTENDED Supports extended ODBC SQL grammar.
ODBC_VER	string	The version of ODBC that the ODBC driver manager supports. This returns a string "MM.mm.rrrr" where *MM* is the major version, *mm* is the minor version, and *rrrr* is the release. The DB2 client always returns "03.01.0000".

Parameters

`connection`
Specifies an active DB2 client connection.

Return Values

Returns an object on a successful call. Returns **FALSE** on failure.

Examples

Example #6 - A db2_client_info() example

To retrieve information about the client, you must pass a valid database connection resource to db2_client_info().

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

```
$client = db2_client_info( $conn );

if ($client) {
   echo "DRIVER_NAME: ";           var_dump( $client->DRIVER_NAME );
   echo "DRIVER_VER: ";            var_dump( $client->DRIVER_VER );
   echo "DATA_SOURCE_NAME: ";      var_dump( $client->DATA_SOURCE_NAME );
   echo "DRIVER_ODBC_VER: ";       var_dump( $client->DRIVER_ODBC_VER );
   echo "ODBC_VER: ";              var_dump( $client->ODBC_VER );
   echo "ODBC_SQL_CONFORMANCE: ";  var_dump( $client->ODBC_SQL_CONFORMANCE
);
   echo "APPL_CODEPAGE: ";         var_dump( $client->APPL_CODEPAGE );
   echo "CONN_CODEPAGE: ";         var_dump( $client->CONN_CODEPAGE );
}
else {
   echo "Error retrieving client information.
    Perhaps your database connection was invalid.";
}
db2_close($conn);

?>
```

The above example will output:

```
DRIVER_NAME: string(8) "libdb2.a"
DRIVER_VER: string(10) "08.02.0001"
DATA_SOURCE_NAME: string(6) "SAMPLE"
DRIVER_ODBC_VER: string(5) "03.51"
ODBC_VER: string(10) "03.01.0000"
ODBC_SQL_CONFORMANCE: string(8) "EXTENDED"
APPL_CODEPAGE: int(819)
CONN_CODEPAGE: int(819)
```

See Also

- db2_server_info()

db2_close

db2_close -- Closes a database connection

Description

bool **db2_close** (resource `$connection`)

This function closes a DB2 client connection created with db2_connect() and returns the corresponding resources to the database server.

If you attempt to close a persistent DB2 client connection created with db2_pconnect(), the close request is ignored and the persistent DB2 client connection remains available for the next caller.

Parameters

`connection`
Specifies an active DB2 client connection.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #7 - Closing a connection

The following example demonstrates a successful attempt to close a connection to an IBM DB2, Cloudscape, or Apache Derby database.

```
<?php
$conn = db2_connect('SAMPLE', 'db2inst1', 'ibmdb2');
$rc = db2_close($conn);
if ($rc) {
   echo "Connection was successfully closed.";
}
?>
```

The above example will output:

```
Connection was successfully closed.
```

See Also

- db2_connect()
- db2_pconnect()

db2_column_privileges

db2_column_privileges -- Returns a result set listing the columns and associated privileges for a table

Description

resource **db2_column_privileges** (resource `$connection` [, string `$qualifier` [, string `$schema` [, string `$table-name` [, string `$column-name`]]]])

Returns a result set listing the columns and associated privileges for a table.

Parameters

`connection`
A valid connection to an IBM DB2, Cloudscape, or Apache Derby database.

`qualifier`
A qualifier for DB2 databases running on OS/390 or z/OS servers. For other databases, pass **NULL** or an empty string.

`schema`
The schema which contains the tables. To match all schemas, pass **NULL** or an empty string.

`table-name`
The name of the table or view. To match all tables in the database, pass **NULL** or an empty string.

`column-name`
The name of the column. To match all columns in the table, pass **NULL** or an empty string.

Return Values

Returns a statement resource with a result set containing rows describing the column privileges for columns matching the specified parameters. The rows are composed of the following columns:

Column name	Description
TABLE_CAT	Name of the catalog. The value is NULL if this table does not have catalogs.
TABLE_SCHEM	Name of the schema.
TABLE_NAME	Name of the table or view.

COLUMN_NAME	Name of the column.
GRANTOR	Authorization ID of the user who granted the privilege.
GRANTEE	Authorization ID of the user to whom the privilege was granted.
PRIVILEGE	The privilege for the column.
IS_GRANTABLE	Whether the GRANTEE is permitted to grant this privilege to other users.

See Also

- db2_columns()
- db2_foreign_keys()
- db2_primary_keys()
- db2_procedure_columns()
- db2_procedures()
- db2_special_columns()
- db2_statistics()
- db2_table_privileges()
- db2_tables()

db2_columns

db2_columns -- Returns a result set listing the columns and associated metadata for a table

Description

resource **db2_columns** (resource `$connection` [, string `$qualifier` [, string `$schema` [, string `$table-name` [, string `$column-name`]]]])

Returns a result set listing the columns and associated metadata for a table.

Parameters

`connection`
A valid connection to an IBM DB2, Cloudscape, or Apache Derby database.

`qualifier`
A qualifier for DB2 databases running on OS/390 or z/OS servers. For other databases, pass **NULL** or an empty string.

`schema`
The schema which contains the tables. To match all schemas, pass *'%'*.

`table-name`
The name of the table or view. To match all tables in the database, pass **NULL** or an empty string.

`column-name`
The name of the column. To match all columns in the table, pass **NULL** or an empty string.

Return Values

Returns a statement resource with a result set containing rows describing the columns matching the specified parameters. The rows are composed of the following columns:

Column name	Description
TABLE_CAT	Name of the catalog. The value is NULL if this table does not have catalogs.
TABLE_SCHEM	Name of the schema.
TABLE_NAME	Name of the table or view.
COLUMN_NAME	Name of the column.

DATA_TYPE	The SQL data type for the column represented as an integer value.
TYPE_NAME	A string representing the data type for the column.
COLUMN_SIZE	An integer value representing the size of the column.
BUFFER_LENGTH	Maximum number of bytes necessary to store data from this column.
DECIMAL_DIGITS	The scale of the column, or **NULL** where scale is not applicable.
NUM_PREC_RADIX	An integer value of either *10* (representing an exact numeric data type), *2* (representing an approximate numeric data type), or **NULL** (representing a data type for which radix is not applicable).
NULLABLE	An integer value representing whether the column is nullable or not.
REMARKS	Description of the column.
COLUMN_DEF	Default value for the column.
SQL_DATA_TYPE	An integer value representing the size of the column.
SQL_DATETIME_SUB	Returns an integer value representing a datetime subtype code, or **NULL** for SQL data types to which this does not apply.
CHAR_OCTET_LENGTH	Maximum length in octets for a character data type column, which matches COLUMN_SIZE for single-byte character set data, or **NULL** for non-character data types.
ORDINAL_POSITION	The 1-indexed position of the column in the table.
IS_NULLABLE	A string value where 'YES' means that the column is nullable and 'NO' means that the column is not nullable.

See Also

- db2_column_privileges()
- db2_foreign_keys()
- db2_primary_keys()
- db2_procedure_columns()
- db2_procedures()
- db2_special_columns()
- db2_statistics()
- db2_table_privileges()
- db2_tables()

db2_commit

db2_commit -- Commits a transaction

Description

bool **db2_commit** (resource `$connection`)

Commits an in-progress transaction on the specified connection resource and begins a new transaction. PHP applications normally default to AUTOCOMMIT mode, so db2_commit() is not necessary unless AUTOCOMMIT has been turned off for the connection resource.

Note
If the specified connection resource is a persistent connection, all transactions in progress for all applications using that persistent connection will be committed. For this reason, persistent connections are not recommended for use in applications that require transactions.

Parameters

`connection`
: A valid database connection resource variable as returned from db2_connect() or db2_pconnect().

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- db2_autocommit()
- db2_rollback()

db2_conn_error

db2_conn_error -- Returns a string containing the SQLSTATE returned by the last connection attempt

Description

string **db2_conn_error** ([resource `$connection`])

db2_conn_error() returns an SQLSTATE value representing the reason the last attempt to connect to a database failed. As db2_connect() returns **FALSE** in the event of a failed connection attempt, you do not pass any parameters to db2_conn_error() to retrieve the SQLSTATE value.

If, however, the connection was successful but becomes invalid over time, you can pass the *`connection`* parameter to retrieve the SQLSTATE value for a specific connection.

To learn what the SQLSTATE value means, you can issue the following command at a DB2 Command Line Processor prompt: **db2 '?** *`sqlstate-value`* **'**. You can also call db2_conn_errormsg() to retrieve an explicit error message and the associated SQLCODE value.

Parameters

`connection`
A connection resource associated with a connection that initially succeeded, but which over time became invalid.

Return Values

Returns the SQLSTATE value resulting from a failed connection attempt. Returns an empty string if there is no error associated with the last connection attempt.

Examples

Example #8 - Retrieving an SQLSTATE value for a failed connection attempt

The following example demonstrates how to return an SQLSTATE value after deliberately passing invalid parameters to db2_connect().

```
<?php
$conn = db2_connect('badname', 'baduser', 'badpassword');
if (!$conn) {
   print "SQLSTATE value: " . db2_conn_error();
}
```

```
?>
```

The above example will output:

```
SQLSTATE value: 08001
```

See Also

- db2_conn_errormsg()
- db2_connect()
- db2_stmt_error()
- db2_stmt_errormsg()

db2_conn_errormsg

db2_conn_errormsg -- Returns the last connection error message and SQLCODE value

Description

string **db2_conn_errormsg** ([resource `$connection`])

db2_conn_errormsg() returns an error message and SQLCODE value representing the reason the last database connection attempt failed. As db2_connect() returns **FALSE** in the event of a failed connection attempt, do not pass any parameters to db2_conn_errormsg() to retrieve the associated error message and SQLCODE value.

If, however, the connection was successful but becomes invalid over time, you can pass the *`connection`* parameter to retrieve the associated error message and SQLCODE value for a specific connection.

Parameters

`connection`
: A connection resource associated with a connection that initially succeeded, but which over time became invalid.

Return Values

Returns a string containing the error message and SQLCODE value resulting from a failed connection attempt. If there is no error associated with the last connection attempt, db2_conn_errormsg() returns an empty string.

Examples

Example #9 - Retrieving the error message returned by a failed connection attempt

The following example demonstrates how to return an error message and SQLCODE value after deliberately passing invalid parameters to db2_connect().

```
<?php
$conn = db2_connect('badname', 'baduser', 'badpassword');
if (!$conn) {
   print db2_conn_errormsg();
}
?>
```

The above example will output:

```
[IBM][CLI Driver] SQL1013N  The database alias name
or database name "BADNAME" could not be found.  SQLSTATE=42705
SQLCODE=-1013
```

See Also

- db2_conn_error()
- db2_connect()
- db2_stmt_error()
- db2_stmt_errormsg()

db2_connect

db2_connect -- Returns a connection to a database

Description

resource **db2_connect** (string `$database`, string `$username`, string `$password` [, array `$options`])

Creates a new connection to an IBM DB2 Universal Database, IBM Cloudscape, or Apache Derby database.

Parameters

`database`
: For a cataloged connection to a database, *`database`* represents the database alias in the DB2 client catalog. For an uncataloged connection to a database, *`database`* represents a complete connection string in the following format: `DATABASE=` *`database`* `;HOSTNAME=` *`hostname`*`;PORT=` *`port`*`;PROTOCOL=TCPIP;UID=` *`username`*`;PWD=` *`password`*`;` where the parameters represent the following values:

  `database`
  : The name of the database.

  `hostname`
  : The hostname or IP address of the database server.

  `port`
  : The TCP/IP port on which the database is listening for requests.

  `username`
  : The username with which you are connecting to the database.

  `password`
  : The password with which you are connecting to the database.

`username`
: The username with which you are connecting to the database. For uncataloged connections, you must pass a **NULL** value or empty string.

`password`
: The password with which you are connecting to the database. For uncataloged connections, you must pass a **NULL** value or empty string.

`options`
: An associative array of connection options that affect the behavior of the connection, where valid array keys include:

  `autocommit`

Passing the *DB2_AUTOCOMMIT_ON* value turns autocommit on for this connection handle. Passing the *DB2_AUTOCOMMIT_OFF* value turns autocommit off for this connection handle.

DB2_ATTR_CASE
Passing the *DB2_CASE_NATURAL* value specifies that column names are returned in natural case. Passing the *DB2_CASE_LOWER* value specifies that column names are returned in lower case. Passing the *DB2_CASE_UPPER* value specifies that column names are returned in upper case.

CURSOR
Passing the *DB2_FORWARD_ONLY* value specifies a forward-only cursor for a statement resource. This is the default cursor type and is supported on all database servers. Passing the *DB2_SCROLLABLE* value specifies a scrollable cursor for a statement resource. This mode enables random access to rows in a result set, but currently is supported only by IBM DB2 Universal Database.

The following new i5/OS options are available as of ibm_db2 version 1.5.1. Note: prior versions of ibm_db2 do not support these new i5 options.

i5_lib
A character value that indicates the default library that will be used for resolving unqualified file references. This is not valid if the connection is using system naming mode.

i5_naming
DB2_I5_NAMING_ON value turns on DB2 UDB CLI iSeries system naming mode. Files are qualified using the slash (/) delimiter. Unqualified files are resolved using the library list for the job. *DB2_I5_NAMING_OFF* value turns off DB2 UDB CLI default naming mode, which is SQL naming. Files are qualified using the period (.) delimiter. Unqualified files are resolved using either the default library or the current user ID.

i5_commit
The *i5_commit* attribute should be set before the db2_connect(). If the value is changed after the connection has been established, and the connection is to a remote data source, the change does not take effect until the next successful db2_connect() for the connection handle. Note: php.ini setting *ibm_db2.i5_allow_commit* ==0 or *DB2_I5_TXN_NO_COMMIT* is the default, but may be overridden with the *i5_commit* option. *DB2_I5_TXN_NO_COMMIT* - Commitment control is not used. *DB2_I5_TXN_READ_UNCOMMITTED* - Dirty reads, nonrepeatable reads, and phantoms are possible. *DB2_I5_TXN_READ_COMMITTED* - Dirty reads are not possible. Nonrepeatable reads, and phantoms are possible. *DB2_I5_TXN_REPEATABLE_READ* - Dirty reads and nonrepeatable reads are not possible. Phantoms are possible. *DB2_I5_TXN_SERIALIZABLE* - Transactions are serializable. Dirty reads, non-repeatable reads, and phantoms are not possible

i5_query_optimize
DB2_FIRST_IO All queries are optimized with the goal of returning the first page of output as fast as possible. This goal works well when the output is controlled by a user who is most likely to cancel the query after viewing the first page of output data. Queries coded with an OPTIMIZE FOR nnn ROWS clause honor the goal specified by the clause. *DB2_ALL_IO* All queries are optimized with the goal of

running the entire query to completion in the shortest amount of elapsed time. This is a good option when the output of a query is being written to a file or report, or the interface is queuing the output data. Queries coded with an OPTIMIZE FOR nnn ROWS clause honor the goal specified by the clause. This is the default.

`i5_dbcs_alloc`
DB2_I5_DBCS_ALLOC_ON value turns on DB2 6X allocation scheme for DBCS translation column size growth. *DB2_I5_DBCS_ALLOC_OFF* value turns off DB2 6X allocation scheme for DBCS translation column size growth. Note: php.ini setting `ibm_db2.i5_dbcs_alloc` ==0 or *DB2_I5_DBCS_ALLOC_OFF* is the default, but may be overridden with the `i5_dbcs_alloc` option.

`i5_date_fmt`
SQL_FMT_ISO - The International Organization for Standardization (ISO) date format yyyy-mm-dd is used. This is the default. *DB2_I5_FMT_USA* - The United States date format mm/dd/yyyy is used. *DB2_I5_FMT_EUR* - The European date format dd.mm.yyyy is used. *DB2_I5_FMT_JIS* - The Japanese Industrial Standard date format yyyy-mm-dd is used. *DB2_I5_FMT_MDY* - The date format mm/dd/yyyy is used. *DB2_I5_FMT_DMY* - The date format dd/mm/yyyy is used. *DB2_I5_FMT_YMD* - The date format yy/mm/dd is used. *DB2_I5_FMT_JUL* - The Julian date format yy/ddd is used. *DB2_I5_FMT_JOB* - The job default is used.

`i5_date_sep`
DB2_I5_SEP_SLASH - A slash (/) is used as the date separator. This is the default. *DB2_I5_SEP_DASH* - A dash (-) is used as the date separator. *DB2_I5_SEP_PERIOD* - A period (.) is used as the date separator. *DB2_I5_SEP_COMMA* - A comma (,) is used as the date separator. *DB2_I5_SEP_BLANK* - A blank is used as the date separator. *DB2_I5_SEP_JOB* - The job default is used

`i5_time_fmt`
DB2_I5_FMT_ISO - The International Organization for Standardization (ISO) time format hh.mm.ss is used. This is the default. *DB2_I5_FMT_USA* - The United States time format hh:mmxx is used, where xx is AM or PM. *DB2_I5_FMT_EUR* - The European time format hh.mm.ss is used. *DB2_I5_FMT_JIS* - The Japanese Industrial Standard time format hh:mm:ss is used. *DB2_I5_FMT_HMS* - The hh:mm:ss format is used.

`i5_time_sep`
DB2_I5_SEP_COLON - A colon (:) is used as the time separator. This is the default. *DB2_I5_SEP_PERIOD* - A period (.) is used as the time separator. *DB2_I5_SEP_COMMA* - A comma (,) is used as the time separator. *DB2_I5_SEP_BLANK* - A blank is used as the time separator. *DB2_I5_SEP_JOB* - The job default is used.

`i5_decimal_sep`
DB2_I5_SEP_PERIOD - A period (.) is used as the decimal separator. This is the default. *DB2_I5_SEP_COMMA* - A comma (,) is used as the decimal separator. *DB2_I5_SEP_JOB* - The job default is used.

Return Values

Returns a connection handle resource if the connection attempt is successful. If the connection attempt fails, db2_connect() returns **FALSE**.

Examples

Example #10 - Creating a cataloged connection

Cataloged connections require you to have previously cataloged the target database through the DB2 Command Line Processor (CLP) or DB2 Configuration Assistant.

```
<?php
$database = 'SAMPLE';
$user = 'db2inst1';
$password = 'ibmdb2';

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

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

The above example will output:

```
Connection succeeded.
```

Example #11 - Creating an uncataloged connection

An uncataloged connection enables you to dynamically connect to a database.

```
<?php
$database = 'SAMPLE';
$user = 'db2inst1';
$password = 'ibmdb2';
$hostname = 'localhost';
$port = 50000;

$conn_string = "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=$database;" .
 "HOSTNAME=$hostname;PORT=$port;PROTOCOL=TCPIP;UID=$user;PWD=$password;";
$conn = db2_connect($conn_string, '', '');

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

```
   echo "Connection failed.";
}
?>
```

The above example will output:

```
Connection succeeded.
```

Example #12 - Creating a connection with autocommit off by default

Passing an array of options to db2_connect() enables you to modify the default behavior of the connection handle.

```
<?php
$database = 'SAMPLE';
$user = 'db2inst1';
$password = 'ibmdb2';
$options = array('autocommit' => DB2_AUTOCOMMIT_OFF);

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

if ($conn) {
   echo "Connection succeeded.\n";
   if (db2_autocommit($conn)) {
        echo "Autocommit is on.\n";
   }
   else {
        echo "Autocommit is off.\n";
   }
   db2_close($conn);
}
else {
   echo "Connection failed.";
}
?>
```

The above example will output:

```
Connection succeeded.
Autocommit is off.
```

Example #13 - i5/OS best performance

To achieve best performance for your i5/OS ibm_db2 1.5.1 PHP application use the default host, userid, and password for your db2_connect().

```
<?php
 $library = "ADC";
 $i5 = db2_connect("", "", "", array("i5_lib"=>"qsys2"));
 $result = db2_exec($i5,
      "select * from systables where table_schema = '$library'");
 while ($row = db2_fetch_both($result)) {
    echo $row['TABLE_NAME']."</br>";
```

```
 }
 db2_close($i5);
?>
```

The above example will output:

```
ANIMALS
NAMES
PICTURES
```

See Also

- db2_close()
- db2_pconnect()

db2_cursor_type

db2_cursor_type -- Returns the cursor type used by a statement resource

Description

int **db2_cursor_type** (resource `$stmt`)

Returns the cursor type used by a statement resource. Use this to determine if you are working with a forward-only cursor or scrollable cursor.

Parameters

`stmt`
: A valid statement resource.

Return Values

Returns either *DB2_FORWARD_ONLY* if the statement resource uses a forward-only cursor or *DB2_SCROLLABLE* if the statement resource uses a scrollable cursor.

See Also

- db2_prepare()

db2_escape_string

db2_escape_string -- Used to escape certain characters

Description

string **db2_escape_string** (string `$string_literal`)

Prepends backslashes to special characters in the string argument.

Parameters

`string_literal`
The string that contains special characters that need to be modified. Characters that are prepended with a backslash are *\x00*, *\n*, *\r*, **, *'*, *"* and *\x1a*.

Return Values

Returns *`string_literal`* with the special characters noted above prepended with backslashes.

Examples

Example #14 - A db2_escape_string() example

Result of using the db2_escape_string() function

```
<?php

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

if ($conn) {
   $str[0] = "All characters: \x00 , \n , \r , \ , ' , \" , \x1a .";
   $str[1] = "Backslash (\). Single quote ('). Double quote (\")";
   $str[2] = "The NULL character \0 must be quoted as well";
   $str[3] = "Intersting characters: \x1a , \x00 .";
   $str[4] = "Nothing to quote";
   $str[5] = 200676;
   $str[6] = "";

   foreach( $str as $string ) {
       echo "db2_escape_string: " . db2_escape_string($string). "\n";
   }
}
?>
```

The above example will output:

```
db2_escape_string: All characters: \0 , \n , \r , \\ , \' , \" , \Z .
db2_escape_string: Backslash (\\). Single quote (\'). Double quote (\")
db2_escape_string: The NULL character \0 must be quoted as well
db2_escape_string: Intersting characters: \Z , \0 .
db2_escape_string: Nothing to quote
db2_escape_string: 200676
db2_escape_string:
```

See Also

- db2_prepare()

db2_exec

db2_exec -- Executes an SQL statement directly

Description

resource **db2_exec** (resource `$connection`, string `$statement` [, array `$options`])

Executes an SQL statement directly.

If you plan to interpolate PHP variables into the SQL statement, understand that this is one of the more common security exposures. Consider calling db2_prepare() to prepare an SQL statement with parameter markers for input values. Then you can call db2_execute() to pass in the input values and avoid SQL injection attacks.

If you plan to repeatedly issue the same SQL statement with different parameters, consider calling db2_prepare() and db2_execute() to enable the database server to reuse its access plan and increase the efficiency of your database access.

Parameters

`connection`
A valid database connection resource variable as returned from db2_connect() or db2_pconnect().

`statement`
An SQL statement. The statement cannot contain any parameter markers.

`options`
An associative array containing statement options. You can use this parameter to request a scrollable cursor on database servers that support this functionality.
`cursor`
Passing the *DB2_FORWARD_ONLY* value requests a forward-only cursor for this SQL statement. This is the default type of cursor, and it is supported by all database servers. It is also much faster than a scrollable cursor. Passing the *DB2_SCROLLABLE* value requests a scrollable cursor for this SQL statement. This type of cursor enables you to fetch rows non-sequentially from the database server. However, it is only supported by DB2 servers, and is much slower than forward-only cursors.

Return Values

Returns a statement resource if the SQL statement was issued successfully, or **FALSE** if the database failed to execute the SQL statement.

Examples

Example #15 - Creating a table with db2_exec()

The following example uses db2_exec() to issue a set of DDL statements in the process of creating a table.

```
<?php
$conn = db2_connect($database, $user, $password);

// Create the test table
$create = 'CREATE TABLE animals (id INTEGER, breed VARCHAR(32),
   name CHAR(16), weight DECIMAL(7,2))';
$result = db2_exec($conn, $create);
if ($result) {
   print "Successfully created the table.\n";
}

// Populate the test table
$animals = array(
   array(0, 'cat', 'Pook', 3.2),
   array(1, 'dog', 'Peaches', 12.3),
   array(2, 'horse', 'Smarty', 350.0),
   array(3, 'gold fish', 'Bubbles', 0.1),
   array(4, 'budgerigar', 'Gizmo', 0.2),
   array(5, 'goat', 'Rickety Ride', 9.7),
   array(6, 'llama', 'Sweater', 150)
);

foreach ($animals as $animal) {
   $rc = db2_exec($conn, "INSERT INTO animals (id, breed, name, weight)
     VALUES ({$animal[0]}, '{$animal[1]}', '{$animal[2]}', {$animal[3]})");
   if ($rc) {
       print "Insert... ";
   }
}
?>
```

The above example will output:

```
Successfully created the table.
Insert... Insert... Insert... Insert... Insert... Insert... Insert...
```

Example #16 - Executing a SELECT statement with a scrollable cursor

The following example demonstrates how to request a scrollable cursor for an SQL statement issued by db2_exec().

```
<?php
$conn = db2_connect($database, $user, $password);
$sql = "SELECT name FROM animals
   WHERE weight < 10.0
   ORDER BY name";
if ($conn) {
   require_once('prepare.inc');
   $stmt = db2_exec($conn, $sql, array('cursor' => DB2_SCROLLABLE));
   while ($row = db2_fetch_array($stmt)) {
```

```
        print "$row[0]\n";
    }
}
?>
```

The above example will output:

```
Bubbles
Gizmo
Pook
Rickety Ride
```

Example #17 - Returning XML data as a SQL ResultSet

The following example demonstrates how to work with documents stored in a XML column using the SAMPLE database. Using some pretty simple SQL/XML, this example returns some of the nodes in a XML document in a SQL ResultSet format that most users are familiar with.

```
<?php

$conn = db2_connect("SAMPLE", "db2inst1", "ibmdb2");

$query = 'SELECT * FROM XMLTABLE(
   XMLNAMESPACES (DEFAULT \'http://posample.org\'),
   \'db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo\'
   COLUMNS
   "CID" VARCHAR (50) PATH \'@Cid\',
   "NAME" VARCHAR (50) PATH \'name\',
   "PHONE" VARCHAR (50) PATH \'phone [ @type = "work"]\'
   ) AS T
   WHERE NAME = \'Kathy Smith\'
   ';
$stmt = db2_exec($conn, $query);

while($row = db2_fetch_object($stmt)){
   printf("$row->CID     $row->NAME     $row->PHONE\n");
}
db2_close($conn);

?>
```

The above example will output:

```
1000     Kathy Smith     416-555-1358
1001     Kathy Smith     905-555-7258
```

Example #18 - Performing a "JOIN" with XML data

The following example works with documents stored in 2 different XML columns in the SAMPLE database. It creates 2 temporary tables from the XML documents from 2 different columns and returns a SQL ResultSet with information regarding shipping status for the customer.

```
<?php

$conn = db2_connect("SAMPLE", "db2inst1", "ibmdb2");

$query = '
   SELECT A.CID, A.NAME, A.PHONE, C.PONUM, C.STATUS
   FROM
   XMLTABLE(
   XMLNAMESPACES (DEFAULT \'http://posample.org\'),
   \'db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo\'
   COLUMNS
   "CID" BIGINT PATH \'@Cid\',
   "NAME" VARCHAR (50) PATH \'name\',
   "PHONE" VARCHAR (50) PATH \'phone [ @type = "work"]\'
   ) as A,
   PURCHASEORDER AS B,
   XMLTABLE (
   XMLNAMESPACES (DEFAULT \'http://posample.org\'),
   \'db2-fn:xmlcolumn("PURCHASEORDER.PORDER")/PurchaseOrder\'
   COLUMNS
   "PONUM"  BIGINT PATH \'@PoNum\',
   "STATUS" VARCHAR (50) PATH \'@Status\'
   ) as C
   WHERE A.CID = B.CUSTID AND
   B.POID = C.PONUM AND
   A.NAME = \'Kathy Smith\'
';

$stmt = db2_exec($conn, $query);

while($row = db2_fetch_object($stmt)){
   printf("$row->CID     $row->NAME     $row->PHONE     $row->PONUM
$row->STATUS\n");
}

db2_close($conn);

?>
```

The above example will output:

```
1001     Kathy Smith     905-555-7258     5002     Shipped
```

Example #19 - Returning SQL data as part of a larger XML document

The following example works with a portion of the PRODUCT.DESCRIPTION documents in the SAMPLE database. It creates a XML document containing product description (XML data) and pricing info (SQL data).

```
<?php

$conn = db2_connect("SAMPLE", "db2inst1", "ibmdb2");

$query = '
SELECT
XMLSERIALIZE(
XMLQUERY(\'
```

```
   declare boundary-space strip;
   declare default element namespace "http://posample.org";
   <promoList> {
   for $prod in $doc/product
   where $prod/description/price < 10.00
   order by $prod/description/price ascending
   return(
       <promoitem> {
       $prod,
       <startdate> {$start} </startdate>,
       <enddate> {$end} </enddate>,
       <promoprice> {$promo} </promoprice>
       } </promoitem>
   )
   } </promoList>
\' passing by ref DESCRIPTION AS "doc",
PROMOSTART as "start",
PROMOEND as "end",
PROMOPRICE as "promo"
RETURNING SEQUENCE)
AS CLOB (32000))
AS NEW_PRODUCT_INFO
FROM PRODUCT
WHERE PID = \'100-100-01\'
';

$stmt = db2_exec($conn, $query);

while($row = db2_fetch_array($stmt)){
   printf("$row[0]\n");
}
db2_close($conn);

?>
```

The above example will output:

```
<promoList xmlns="http://posample.org">
   <promoitem>
   <product pid="100-100-01">
       <description>
           <name>Snow Shovel, Basic 22 inch</name>
           <details>Basic Snow Shovel, 22 inches wide, straight handle with
D-Grip</details>
           <price>9.99</price>
           <weight>1 kg</weight>
       </description>
   </product>
   <startdate>2004-11-19</startdate>
   <enddate>2004-12-19</enddate>
   <promoprice>7.25</promoprice>
   </promoitem>
</promoList>
```

See Also

- db2_execute()
- db2_prepare()

db2_execute

db2_execute -- Executes a prepared SQL statement

Description

bool **db2_execute** (resource `$stmt` [, array `$parameters`])

db2_execute() executes an SQL statement that was prepared by db2_prepare().

If the SQL statement returns a result set, for example, a SELECT statement or a CALL to a stored procedure that returns one or more result sets, you can retrieve a row as an array from the *stmt* resource using db2_fetch_assoc(), db2_fetch_both(), or db2_fetch_array(). Alternatively, you can use db2_fetch_row() to move the result set pointer to the next row and fetch a column at a time from that row with db2_result().

Refer to db2_prepare() for a brief discussion of the advantages of using db2_prepare() and db2_execute() rather than db2_exec().

Parameters

`stmt`
A prepared statement returned from db2_prepare().

`parameters`
An array of input parameters matching any parameter markers contained in the prepared statement.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #20 - Preparing and executing an SQL statement with parameter markers

The following example prepares an INSERT statement that accepts four parameter markers, then iterates over an array of arrays containing the input values to be passed to db2_execute().

```
<?php
$pet = array(0, 'cat', 'Pook', 3.2);

$insert = 'INSERT INTO animals (id, breed, name, weight)
```

```
   VALUES (?, ?, ?, ?)';

$stmt = db2_prepare($conn, $insert);
if ($stmt) {
   $result = db2_execute($stmt, $pet);
   if ($result) {
       print "Successfully added new pet.";
   }
}
?>
```

The above example will output:

```
Successfully added new pet.
```

Example #21 - Calling a stored procedure with an OUT parameter

The following example prepares a CALL statement that accepts one parameter marker representing an OUT parameter, binds the PHP variable *$my_pets* to the parameter using db2_bind_param(), then issues db2_execute() to execute the CALL statement. After the CALL to the stored procedure has been made, the value of *$num_pets* changes to reflect the value returned by the stored procedure for that OUT parameter.

```
<?php
$num_pets = 0;
$res = db2_prepare($conn, "CALL count_my_pets(?)");
$rc = db2_bind_param($res, 1, "num_pets", DB2_PARAM_OUT);
$rc = db2_execute($res);
print "I have $num_pets pets!";
?>
```

The above example will output:

```
I have 7 pets!
```

Example #22 - Returning XML data as a SQL ResultSet

The following example demonstrates how to work with documents stored in a XML column using the SAMPLE database. Using some pretty simple SQL/XML, this example returns some of the nodes in a XML document in a SQL ResultSet format that most users are familiar with.

```
<?php

$conn = db2_connect("SAMPLE", "db2inst1", "ibmdb2");

$query = 'SELECT * FROM XMLTABLE(
   XMLNAMESPACES (DEFAULT \'http://posample.org\'),
   \'db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo\'
   COLUMNS
   "CID" VARCHAR (50) PATH \'@Cid\',
   "NAME" VARCHAR (50) PATH \'name\',
   "PHONE" VARCHAR (50) PATH \'phone [ @type = "work"]\'
```

```
   ) AS T
   WHERE NAME = ?
   ';

$stmt = db2_prepare($conn, $query);

$name = 'Kathy Smith';

if ($stmt) {
   db2_bind_param($stmt, 1, "name", DB2_PARAM_IN);
   db2_execute($stmt);

   while($row = db2_fetch_object($stmt)){
   printf("$row->CID     $row->NAME     $row->PHONE\n");
   }
}
db2_close($conn);

?>
```

The above example will output:

```
1000     Kathy Smith     416-555-1358
1001     Kathy Smith     905-555-7258
```

Example #23 - Performing a "JOIN" with XML data

The following example works with documents stored in 2 different XML columns in the SAMPLE database. It creates 2 temporary tables from the XML documents from 2 different columns and returns a SQL ResultSet with information regarding shipping status for the customer.

```
<?php

$conn = db2_connect("SAMPLE", "db2inst1", "ibmdb2");

$query = '
SELECT A.CID, A.NAME, A.PHONE, C.PONUM, C.STATUS
FROM
XMLTABLE(
XMLNAMESPACES (DEFAULT \'http://posample.org\'),
\'db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo\'
COLUMNS
"CID" BIGINT PATH \'@Cid\',
"NAME" VARCHAR (50) PATH \'name\',
"PHONE" VARCHAR (50) PATH \'phone [ @type = "work"]\'
) as A,
PURCHASEORDER AS B,
XMLTABLE (
XMLNAMESPACES (DEFAULT \'http://posample.org\'),
\'db2-fn:xmlcolumn("PURCHASEORDER.PORDER")/PurchaseOrder\'
COLUMNS
"PONUM"  BIGINT PATH \'@PoNum\',
"STATUS" VARCHAR (50) PATH \'@Status\'
) as C
WHERE A.CID = B.CUSTID AND
   B.POID = C.PONUM AND
```

```
   A.NAME = ?
';

$stmt = db2_prepare($conn, $query);

$name = 'Kathy Smith';

if ($stmt) {
   db2_bind_param($stmt, 1, "name", DB2_PARAM_IN);
   db2_execute($stmt);

   while($row = db2_fetch_object($stmt)){
   printf("$row->CID     $row->NAME     $row->PHONE     $row->PONUM
$row->STATUS\n");
   }
}

db2_close($conn);

?>
```

The above example will output:

```
1001     Kathy Smith     905-555-7258     5002     Shipped
```

Example #24 - Returning SQL data as part of a larger XML document

The following example works with a portion of the PRODUCT.DESCRIPTION documents in the SAMPLE database. It creates a XML document containing product description (XML data) and pricing info (SQL data).

```
<?php

$conn = db2_connect("SAMPLE", "db2inst1", "ibmdb2");

$query = '
SELECT
XMLSERIALIZE(
XMLQUERY(\'
   declare boundary-space strip;
   declare default element namespace "http://posample.org";
   <promoList> {
   for $prod in $doc/product
   where $prod/description/price < 10.00
   order by $prod/description/price ascending
   return(
       <promoitem> {
       $prod,
       <startdate> {$start} </startdate>,
       <enddate> {$end} </enddate>,
       <promoprice> {$promo} </promoprice>
           } </promoitem>
   )
   } </promoList>
\' passing by ref DESCRIPTION AS "doc",
PROMOSTART as "start",
PROMOEND as "end",
```

```
PROMOPRICE as "promo"
RETURNING SEQUENCE)
AS CLOB (32000))
AS NEW_PRODUCT_INFO
FROM PRODUCT
WHERE PID = ?
';

$stmt = db2_prepare($conn, $query);

$pid = "100-100-01";

if ($stmt) {
   db2_bind_param($stmt, 1, "pid", DB2_PARAM_IN);
   db2_execute($stmt);

   while($row = db2_fetch_array($stmt)){
   printf("$row[0]\n");
   }
}

db2_close($conn);

?>
```

The above example will output:

```
<promoList xmlns="http://posample.org">
   <promoitem>
   <product pid="100-100-01">
       <description>
           <name>Snow Shovel, Basic 22 inch</name>
           <details>Basic Snow Shovel, 22 inches wide, straight handle with
D-Grip</details>
           <price>9.99</price>
           <weight>1 kg</weight>
       </description>
   </product>
   <startdate>2004-11-19</startdate>
   <enddate>2004-12-19</enddate>
   <promoprice>7.25</promoprice>
   </promoitem>
</promoList>
```

See Also

- db2_exec()
- db2_fetch_array()
- db2_fetch_assoc()
- db2_fetch_both()
- db2_fetch_row()
- db2_prepare()
- db2_result()

db2_fetch_array

db2_fetch_array -- Returns an array, indexed by column position, representing a row in a result set

Description

array **db2_fetch_array** (resource `$stmt` [, int `$row_number`])

Returns an array, indexed by column position, representing a row in a result set. The columns are 0-indexed.

Parameters

`stmt`
A valid *stmt* resource containing a result set.

`row_number`
Requests a specific 1-indexed row from the result set. Passing this parameter results in a PHP warning if the result set uses a forward-only cursor.

Return Values

Returns a 0-indexed array with column values indexed by the column position representing the next or requested row in the result set. Returns **FALSE** if there are no rows left in the result set, or if the row requested by *`row_number`* does not exist in the result set.

Examples

Example #25 - Iterating through a forward-only cursor

If you call db2_fetch_array() without a specific row number, it automatically retrieves the next row in the result set.

```
<?php

$sql = "SELECT id, name, breed, weight FROM animals ORDER BY breed";
$stmt = db2_prepare($conn, $sql);
$result = db2_execute($stmt);

while ($row = db2_fetch_array($stmt)) {
   printf ("%-5d %-16s %-32s %10s\n",
       $row[0], $row[1], $row[2], $row[3]);
}
?>
```

The above example will output:

```
0     Pook            cat                                3.20
5     Rickety Ride    goat                               9.70
2     Smarty          horse                            350.00
```

Example #26 - Retrieving specific rows with db2_fetch_array() from a scrollable cursor

If your result set uses a scrollable cursor, you can call db2_fetch_array() with a specific row number. The following example retrieves every other row in the result set, starting with the second row.

```
<?php

$sql = "SELECT id, name, breed, weight FROM animals ORDER BY breed";
$result = db2_exec($stmt, $sql, array('cursor' => DB2_SCROLLABLE));

$i=2;
while ($row = db2_fetch_array($result, $i)) {
   printf ("%-5d %-16s %-32s %10s\n",
       $row[0], $row[1], $row[2], $row[3]);
   $i = $i + 2;
}
?>
```

The above example will output:

```
0     Pook            cat                                3.20
5     Rickety Ride    goat                               9.70
2     Smarty          horse                            350.00
```

See Also

- db2_fetch_assoc()
- db2_fetch_both()
- db2_fetch_object()
- db2_fetch_row()
- db2_result()

db2_fetch_assoc

db2_fetch_assoc -- Returns an array, indexed by column name, representing a row in a result set

Description

array **db2_fetch_assoc** (resource `$stmt` [, int `$row_number`])

Returns an array, indexed by column name, representing a row in a result set.

Parameters

`stmt`
A valid *stmt* resource containing a result set.

`row_number`
Requests a specific 1-indexed row from the result set. Passing this parameter results in a PHP warning if the result set uses a forward-only cursor.

Return Values

Returns an associative array with column values indexed by the column name representing the next or requested row in the result set. Returns **FALSE** if there are no rows left in the result set, or if the row requested by *`row_number`* does not exist in the result set.

Examples

Example #27 - Iterating through a forward-only cursor

If you call db2_fetch_assoc() without a specific row number, it automatically retrieves the next row in the result set.

```
<?php

$sql = "SELECT id, name, breed, weight FROM animals ORDER BY breed";
$stmt = db2_prepare($conn, $sql);
$result = db2_execute($stmt);

while ($row = db2_fetch_assoc($stmt)) {
   printf ("%-5d %-16s %-32s %10s\n",
       $row['ID'], $row['NAME'], $row['BREED'], $row['WEIGHT']);
}
?>
```

The above example will output:

```
0     Pook            cat                              3.20
5     Rickety Ride    goat                             9.70
2     Smarty          horse                          350.00
```

Example #28 - Retrieving specific rows with db2_fetch_assoc() from a scrollable cursor

If your result set uses a scrollable cursor, you can call db2_fetch_assoc() with a specific row number. The following example retrieves every other row in the result set, starting with the second row.

```
<?php

$sql = "SELECT id, name, breed, weight FROM animals ORDER BY breed";
$result = db2_exec($stmt, $sql, array('cursor' => DB2_SCROLLABLE));

$i=2;
while ($row = db2_fetch_assoc($result, $i)) {
   printf ("%-5d %-16s %-32s %10s\n",
       $row['ID'], $row['NAME'], $row['BREED'], $row['WEIGHT']);
   $i = $i + 2;
}
?>
```

The above example will output:

```
0     Pook            cat                              3.20
5     Rickety Ride    goat                             9.70
2     Smarty          horse                          350.00
```

See Also

- db2_fetch_array()
- db2_fetch_both()
- db2_fetch_object()
- db2_fetch_row()
- db2_result()

db2_fetch_both

db2_fetch_both -- Returns an array, indexed by both column name and position, representing a row in a result set

Description

array **db2_fetch_both** (resource `$stmt` [, int `$row_number`])

Returns an array, indexed by both column name and position, representing a row in a result set. Note that the row returned by db2_fetch_both() requires more memory than the single-indexed arrays returned by db2_fetch_assoc() or db2_fetch_array().

Parameters

`stmt`
A valid *stmt* resource containing a result set.

`row_number`
Requests a specific 1-indexed row from the result set. Passing this parameter results in a PHP warning if the result set uses a forward-only cursor.

Return Values

Returns an associative array with column values indexed by both the column name and 0-indexed column number. The array represents the next or requested row in the result set. Returns **FALSE** if there are no rows left in the result set, or if the row requested by *`row_number`* does not exist in the result set.

Examples

Example #29 - Iterating through a forward-only cursor

If you call db2_fetch_both() without a specific row number, it automatically retrieves the next row in the result set. The following example accesses columns in the returned array by both column name and by numeric index.

```
<?php

$sql = "SELECT id, name, breed, weight FROM animals ORDER BY breed";
$stmt = db2_prepare($conn, $sql);
$result = db2_execute($stmt);

while ($row = db2_fetch_both($stmt)) {
   printf ("%-5d %-16s %-32s %10s\n",
       $row['ID'], $row[0], $row['BREED'], $row[3]);
```

```
}
?>
```

The above example will output:

```
0     Pook            cat                                3.20
5     Rickety Ride    goat                               9.70
2     Smarty          horse                            350.00
```

Example #30 - Retrieving specific rows with db2_fetch_both() from a scrollable cursor

If your result set uses a scrollable cursor, you can call db2_fetch_both() with a specific row number. The following example retrieves every other row in the result set, starting with the second row.

```
<?php

$sql = "SELECT id, name, breed, weight FROM animals ORDER BY breed";
$result = db2_exec($stmt, $sql, array('cursor' => DB2_SCROLLABLE));

$i=2;
while ($row = db2_fetch_both($result, $i)) {
   printf ("%-5d %-16s %-32s %10s\n",
       $row[0], $row['NAME'], $row[2], $row['WEIGHT']);
   $i = $i + 2;
}
?>
```

The above example will output:

```
0     Pook            cat                                3.20
5     Rickety Ride    goat                               9.70
2     Smarty          horse                            350.00
```

See Also

- db2_fetch_array()
- db2_fetch_assoc()
- db2_fetch_object()
- db2_fetch_row()
- db2_result()

db2_fetch_object

db2_fetch_object -- Returns an object with properties representing columns in the fetched row

Description

object **db2_fetch_object** (resource `$stmt` [, int `$row_number`])

Returns an object in which each property represents a column returned in the row fetched from a result set.

Parameters

`stmt`
A valid *stmt* resource containing a result set.

`row_number`
Requests a specific 1-indexed row from the result set. Passing this parameter results in a PHP warning if the result set uses a forward-only cursor.

Return Values

Returns an object representing a single row in the result set. The properties of the object map to the names of the columns in the result set.

The IBM DB2, Cloudscape, and Apache Derby database servers typically fold column names to upper-case, so the object properties will reflect that case.

If your SELECT statement calls a scalar function to modify the value of a column, the database servers return the column number as the name of the column in the result set. If you prefer a more descriptive column name and object property, you can use the AS clause to assign a name to the column in the result set.

Returns **FALSE** if no row was retrieved.

Examples

Example #31 - A db2_fetch_object() example

The following example issues a SELECT statement with a scalar function, RTRIM, that removes whitespace from the end of the column. Rather than creating an object with the properties "BREED" and "2", we use the AS clause in the SELECT statement to assign the name "name" to the modified column. The database server folds the column

names to upper-case, resulting in an object with the properties "BREED" and "NAME".

```
<?php
$conn = db2_connect($database, $user, $password);

$sql = "SELECT breed, RTRIM(name) AS name
   FROM animals
   WHERE id = ?";

if ($conn) {
   $stmt = db2_prepare($conn, $sql);
   db2_execute($stmt, array(0));

   while ($pet = db2_fetch_object($stmt)) {
       echo "Come here, {$pet->NAME}, my little {$pet->BREED}!";
   }
   db2_close($conn);
}
?>
```

The above example will output:

```
Come here, Pook, my little cat!
```

See Also

- db2_fetch_array()
- db2_fetch_assoc()
- db2_fetch_both()
- db2_fetch_row()
- db2_result()

db2_fetch_row

db2_fetch_row -- Sets the result set pointer to the next row or requested row

Description

bool **db2_fetch_row** (resource `$stmt` [, int `$row_number`])

Use db2_fetch_row() to iterate through a result set, or to point to a specific row in a result set if you requested a scrollable cursor.

To retrieve individual fields from the result set, call the db2_result() function.

Rather than calling db2_fetch_row() and db2_result(), most applications will call one of db2_fetch_assoc(), db2_fetch_both(), or db2_fetch_array() to advance the result set pointer and return a complete row as an array.

Parameters

`stmt`
A valid *stmt* resource.

`row_number`
With scrollable cursors, you can request a specific row number in the result set. Row numbering is 1-indexed.

Return Values

Returns **TRUE** if the requested row exists in the result set. Returns **FALSE** if the requested row does not exist in the result set.

Examples

Example #32 - Iterating through a result set

The following example demonstrates how to iterate through a result set with db2_fetch_row() and retrieve columns from the result set with db2_result().

```
<?php
$sql = 'SELECT name, breed FROM animals WHERE weight < ?';
$stmt = db2_prepare($conn, $sql);
db2_execute($stmt, array(10));
while (db2_fetch_row($stmt)) {
   $name = db2_result($stmt, 0);
   $breed = db2_result($stmt, 1);
   print "$name $breed";
```

```
}
?>
```

The above example will output:

```
cat Pook
gold fish Bubbles
budgerigar Gizmo
goat Rickety Ride
```

Example #33 - i5/OS recommended alternatives to db2_fetch_row/db2_result

On i5/OS it is recommended that you use db2_fetch_both(), db2_fetch_array(), or db2_fetch_object() over db2_fetch_row() / db2_result(). In general db2_fetch_row() / db2_result() have more issues with various column types in *EBCIDIC* to *ASCII* translation, including possible truncation in *DBCS* applications. You may also find the performance of db2_fetch_both(), db2_fetch_array(), and db2_fetch_object() to be superior to db2_fetch_row() / db2_result().

```
<?php
 $conn = db2_connect("","","");
 $sql = 'SELECT SPECIFIC_SCHEMA, SPECIFIC_NAME, ROUTINE_SCHEMA,
ROUTINE_NAME, ROUTINE_TYPE, ROUTINE_CREATED, ROUTINE_BODY, IN_PARMS,
OUT_PARMS, INOUT_PARMS, PARAMETER_STYLE, EXTERNAL_NAME, EXTERNAL_LANGUAGE
FROM QSYS2.SYSROUTINES FETCH FIRST 2 ROWS ONLY';
 $stmt = db2_exec($conn, $sql, array('cursor' => DB2_SCROLLABLE));
 while ($row = db2_fetch_both($stmt)){
   echo "<br>db2_fetch_both {$row['SPECIFIC_NAME']}
{$row['ROUTINE_CREATED']} {$row[5]}";
 }
 $stmt = db2_exec($conn, $sql, array('cursor' => DB2_SCROLLABLE));
 while ($row = db2_fetch_array($stmt)){
   echo "<br>db2_fetch_array {$row[1]}  {$row[5]}";
 }
 $stmt = db2_exec($conn, $sql, array('cursor' => DB2_SCROLLABLE));
 while ($row = db2_fetch_object($stmt)){
   echo "<br>db2_fetch_object {$row->SPECIFIC_NAME}
{$row->ROUTINE_CREATED}";
 }
 db2_close($conn);
?>
```

The above example will output:

```
db2_fetch_both MATCH_ANIMAL 2006-08-25-17.10.23.775000
2006-08-25-17.10.23.775000
db2_fetch_both MULTIRESULTS 2006-10-17-10.11.05.308000
2006-10-17-10.11.05.308000
db2_fetch_array MATCH_ANIMAL 2006-08-25-17.10.23.775000
db2_fetch_array MULTIRESULTS 2006-10-17-10.11.05.308000
db2_fetch_object MATCH_ANIMAL 2006-08-25-17.10.23.775000
db2_fetch_object MULTIRESULTS 2006-10-17-10.11.05.308000
```

See Also

- db2_fetch_array()
- db2_fetch_assoc()
- db2_fetch_both()
- db2_fetch_object()
- db2_result()

db2_field_display_size

db2_field_display_size -- Returns the maximum number of bytes required to display a column

Description

int **db2_field_display_size** (resource `$stmt`, mixed `$column`)

Returns the maximum number of bytes required to display a column in a result set.

Parameters

`stmt`
: Specifies a statement resource containing a result set.

`column`
: Specifies the column in the result set. This can either be an integer representing the 0-indexed position of the column, or a string containing the name of the column.

Return Values

Returns an integer value with the maximum number of bytes required to display the specified column. If the column does not exist in the result set, db2_field_display_size() returns **FALSE**.

See Also

- db2_field_name()
- db2_field_num()
- db2_field_precision()
- db2_field_scale()
- db2_field_type()
- db2_field_width()

db2_field_name

db2_field_name -- Returns the name of the column in the result set

Description

string **db2_field_name** (resource `$stmt`, mixed `$column`)

Returns the name of the specified column in the result set.

Parameters

`stmt`
: Specifies a statement resource containing a result set.

`column`
: Specifies the column in the result set. This can either be an integer representing the 0-indexed position of the column, or a string containing the name of the column.

Return Values

Returns a string containing the name of the specified column. If the specified column does not exist in the result set, db2_field_name() returns **FALSE**.

See Also

- db2_field_display_size()
- db2_field_num()
- db2_field_precision()
- db2_field_scale()
- db2_field_type()
- db2_field_width()

db2_field_num

db2_field_num -- Returns the position of the named column in a result set

Description

int **db2_field_num** (resource `$stmt`, mixed `$column`)

Returns the position of the named column in a result set.

Parameters

`stmt`
: Specifies a statement resource containing a result set.

`column`
: Specifies the column in the result set. This can either be an integer representing the 0-indexed position of the column, or a string containing the name of the column.

Return Values

Returns an integer containing the 0-indexed position of the named column in the result set. If the specified column does not exist in the result set, db2_field_num() returns **FALSE**.

See Also

- db2_field_display_size()
- db2_field_name()
- db2_field_precision()
- db2_field_scale()
- db2_field_type()
- db2_field_width()

db2_field_precision

db2_field_precision -- Returns the precision of the indicated column in a result set

Description

int **db2_field_precision** (resource `$stmt`, mixed `$column`)

Returns the precision of the indicated column in a result set.

Parameters

`stmt`
: Specifies a statement resource containing a result set.

`column`
: Specifies the column in the result set. This can either be an integer representing the 0-indexed position of the column, or a string containing the name of the column.

Return Values

Returns an integer containing the precision of the specified column. If the specified column does not exist in the result set, db2_field_precision() returns **FALSE**.

See Also

- db2_field_display_size()
- db2_field_name()
- db2_field_num()
- db2_field_scale()
- db2_field_type()
- db2_field_width()

db2_field_scale

db2_field_scale -- Returns the scale of the indicated column in a result set

Description

int **db2_field_scale** (resource `$stmt`, mixed `$column`)

Returns the scale of the indicated column in a result set.

Parameters

`stmt`
: Specifies a statement resource containing a result set.

`column`
: Specifies the column in the result set. This can either be an integer representing the 0-indexed position of the column, or a string containing the name of the column.

Return Values

Returns an integer containing the scale of the specified column. If the specified column does not exist in the result set, db2_field_scale() returns **FALSE**.

See Also

- db2_field_display_size()
- db2_field_name()
- db2_field_num()
- db2_field_precision()
- db2_field_type()
- db2_field_width()

db2_field_type

db2_field_type -- Returns the data type of the indicated column in a result set

Description

string **db2_field_type** (resource `$stmt`, mixed `$column`)

Returns the data type of the indicated column in a result set.

Parameters

`stmt`
: Specifies a statement resource containing a result set.

`column`
: Specifies the column in the result set. This can either be an integer representing the 0-indexed position of the column, or a string containing the name of the column.

Return Values

Returns a string containing the defined data type of the specified column. If the specified column does not exist in the result set, db2_field_type() returns **FALSE**.

See Also

- db2_field_display_size()
- db2_field_name()
- db2_field_num()
- db2_field_precision()
- db2_field_scale()
- db2_field_width()

db2_field_width

db2_field_width -- Returns the width of the current value of the indicated column in a result set

Description

int **db2_field_width** (resource `$stmt`, mixed `$column`)

Returns the width of the current value of the indicated column in a result set. This is the maximum width of the column for a fixed-length data type, or the actual width of the column for a variable-length data type.

Parameters

`stmt`
: Specifies a statement resource containing a result set.

`column`
: Specifies the column in the result set. This can either be an integer representing the 0-indexed position of the column, or a string containing the name of the column.

Return Values

Returns an integer containing the width of the specified character or binary data type column in a result set. If the specified column does not exist in the result set, db2_field_width() returns **FALSE**.

See Also

- db2_field_display_size()
- db2_field_name()
- db2_field_num()
- db2_field_precision()
- db2_field_scale()
- db2_field_type()

db2_foreign_keys

db2_foreign_keys -- Returns a result set listing the foreign keys for a table

Description

resource **db2_foreign_keys** (resource `$connection`, string `$qualifier`, string `$schema`, string `$table-name`)

Returns a result set listing the foreign keys for a table.

Parameters

`connection`
A valid connection to an IBM DB2, Cloudscape, or Apache Derby database.

`qualifier`
A qualifier for DB2 databases running on OS/390 or z/OS servers. For other databases, pass **NULL** or an empty string.

`schema`
The schema which contains the tables. If *`schema`* is **NULL**, db2_foreign_keys() matches the schema for the current connection.

`table-name`
The name of the table.

Return Values

Returns a statement resource with a result set containing rows describing the foreign keys for the specified table. The result set is composed of the following columns:

Column name	Description
PKTABLE_CAT	Name of the catalog for the table containing the primary key. The value is NULL if this table does not have catalogs.
PKTABLE_SCHEM	Name of the schema for the table containing the primary key.
PKTABLE_NAME	Name of the table containing the primary key.
PKCOLUMN_NAME	Name of the column containing the primary key.

FKTABLE_CAT	Name of the catalog for the table containing the foreign key. The value is NULL if this table does not have catalogs.
FKTABLE_SCHEM	Name of the schema for the table containing the foreign key.
FKTABLE_NAME	Name of the table containing the foreign key.
FKCOLUMN_NAME	Name of the column containing the foreign key.
KEY_SEQ	1-indexed position of the column in the key.
UPDATE_RULE	Integer value representing the action applied to the foreign key when the SQL operation is UPDATE.
DELETE_RULE	Integer value representing the action applied to the foreign key when the SQL operation is DELETE.
FK_NAME	The name of the foreign key.
PK_NAME	The name of the primary key.
DEFERRABILITY	An integer value representing whether the foreign key deferrability is SQL_INITIALLY_DEFERRED, SQL_INITIALLY_IMMEDIATE, or SQL_NOT_DEFERRABLE.

See Also

- db2_column_privileges()
- db2_columns()
- db2_primary_keys()
- db2_procedure_columns()
- db2_procedures()
- db2_special_columns()
- db2_statistics()
- db2_table_privileges()
- db2_tables()

db2_free_result

db2_free_result -- Frees resources associated with a result set

Description

bool **db2_free_result** (resource `$stmt`)

Frees the system and database resources that are associated with a result set. These resources are freed implicitly when a script finishes, but you can call db2_free_result() to explicitly free the result set resources before the end of the script.

Parameters

`stmt`
A valid statement resource.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- db2_free_stmt()

db2_free_stmt

db2_free_stmt -- Frees resources associated with the indicated statement resource

Description

bool **db2_free_stmt** (resource `$stmt`)

Frees the system and database resources that are associated with a statement resource. These resources are freed implicitly when a script finishes, but you can call db2_free_stmt() to explicitly free the statement resources before the end of the script.

Parameters

`stmt`
A valid statement resource.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- db2_free_result()

db2_get_option

db2_get_option -- Retrieves an option value for a statement resource or a connection resource

Description

string **db2_get_option** (resource `$resource`, string `$option`)

Retrieves the value of a specified option value for a statement resource or a connection resource.

Parameters

`resource`
A valid statement resource as returned from db2_prepare() or a valid connection resource as returned from db2_connect() or db2_pconnect().

`option`
A valid statement or connection options. The following new options are available as of ibm_db2 version 1.6.0. They provide useful tracking information that can be set during execution with db2_get_option().

Note
Note Prior versions of ibm_db2 do not support these new options. When the value in each option is being set, some servers might not handle the entire length provided and might truncate the value. To ensure that the data specified in each option is converted correctly when transmitted to a host system, use only the characters A through Z, 0 through 9, and the underscore (_) or period (.).

`userid`
SQL_ATTR_INFO_USERID - A pointer to a null-terminated character string used to identify the client user ID sent to the host database server when using DB2 Connect.

Note
Note DB2 for z/OS and OS/390 servers support up to a length of 16 characters. This user-id is not to be confused with the authentication user-id, it is for

identification purposes only and is not used for any authorization.

acctstr
SQL_ATTR_INFO_ACCTSTR - A pointer to a null-terminated character string used to identify the client accounting string sent to the host database server when using DB2 Connect.

Note

Note

DB2 for z/OS and OS/390 servers support up to a length of 200 characters.

applname
SQL_ATTR_INFO_APPLNAME - A pointer to a null-terminated character string used to identify the client application name sent to the host database server when using DB2 Connect.

Note

Note

DB2 for z/OS and OS/390 servers support up to a length of 32 characters.

wrkstnname
SQL_ATTR_INFO_WRKSTNNAME - A pointer to a null-terminated character string used to identify the client workstation name sent to the host database server when using DB2 Connect.

Note

Note

DB2 for z/OS and OS/390 servers support up to a length of 18 characters.

The following table specifies which options are compatible with the available resource types:

Resource-Parameter Matrix

Key	Value	Resource Type		

		Connection	Statement	Result Set
userid	*SQL_ATTR_INFO_USERID*	X	X	-
acctstr	*SQL_ATTR_INFO_ACCTSTR*	X	X	-
applname	*SQL_ATTR_INFO_APPLNAME*	X	X	-
wrkstnname	*SQL_ATTR_INFO_WRKSTNNAME*	X	X	-

Return Values

Returns the current setting of the connection attribute provided on success or **FALSE** on failure.

Examples

Example #34 - Setting and retrieving parameters through a connection resource

```
<?php
/* Database Connection Parameters */
$database = 'SAMPLE';
$user     = 'db2inst1';
$password = 'ibmdb2';

/* Obtain Connection Resource */
$conn = db2_connect($database, $user, $password);

echo "Client attributes passed through connection string:\n";

/* Create the associative options array with valid key-value pairs */
/* Assign the attributes through connection string */
/* Access the options specified */
$options1 = array('userid' => 'db2inst1');
$conn1 = db2_connect($database, $user, $password, $options1);
$val = db2_get_option($conn1, 'userid');
echo $val . "\n";

$options2 = array('acctstr' => 'account');
$conn2 = db2_connect($database, $user, $password, $options2);
$val = db2_get_option($conn2, 'acctstr');
echo $val . "\n";

$options3 = array('applname' => 'myapp');
$conn3 = db2_connect($database, $user, $password, $options3);
$val = db2_get_option($conn3, 'applname');
```

```
echo $val . "\n";

$options4 = array('wrkstnname' => 'workstation');
$conn4 = db2_connect($database, $user, $password, $options4);
$val = db2_get_option($conn4, 'wrkstnname');
echo $val . "\n";

echo "Client attributes passed post-connection:\n";

/* Create the associative options array with valid key-value pairs */
/* Assign the attributes after a connection is made */
/* Access the options specified */
$options5 = array('userid' => 'db2inst1');
$conn5 = db2_connect($database, $user, $password);
$rc = db2_set_option($conn5, $options5, 1);
$val = db2_get_option($conn5, 'userid');
echo $val . "\n";

$options6 = array('acctstr' => 'account');
$conn6 = db2_connect($database, $user, $password);
$rc = db2_set_option($conn6, $options6, 1);
$val = db2_get_option($conn6, 'acctstr');
echo $val . "\n";

$options7 = array('applname' => 'myapp');
$conn7 = db2_connect($database, $user, $password);
$rc = db2_set_option($conn7, $options7, 1);
$val = db2_get_option($conn7, 'applname');
echo $val . "\n";

$options8 = array('wrkstnname' => 'workstation');
$conn8 = db2_connect($database, $user, $password);
$rc = db2_set_option($conn8, $options8, 1);
$val = db2_get_option($conn8, 'wrkstnname');
echo $val . "\n";
?>
```

The above example will output:

```
Client attributes passed through connection string:
db2inst1
account
myapp
workstation
Client attributes passed post-connection:
db2inst1
account
myapp
workstation
```

See Also

- db2_connect()
- db2_cursor_type()

- db2_exec()
- db2_set_option()
- db2_pconnect()
- db2_prepare()

db2_lob_read

db2_lob_read -- Gets a user defined size of LOB files with each invocation

Description

string **db2_lob_read** (resource `$stmt`, int `$colnum`, int `$length`)

Use db2_lob_read() to iterate through a specified column of a result set and retrieve a user defined size of LOB data.

Parameters

`stmt`
A valid *stmt* resource containing LOB data.

`colnum`
A valid column number in the result set of the *stmt* resource.

`length`
The size of the LOB data to be retrieved from the *stmt* resource.

Return Values

Returns the amount of data the user specifies. Returns **FALSE** if the data cannot be retrieved.

Examples

Example #35 - Iterating through different types of data

```
<?php

/* Database Connection Parameters */
$db = 'SAMPLE';
$username = 'db2inst1';
$password = 'ibmdb2';

/* Obtain Connection Resource */
$conn = db2_connect($db,$username,$password);

if ($conn) {
   $drop = 'DROP TABLE clob_stream';
   $result = @db2_exec( $conn, $drop );
```

```
   $create = 'CREATE TABLE clob_stream (id INTEGER, my_clob CLOB)';
   $result = db2_exec( $conn, $create );

   $variable = "";
   $stmt = db2_prepare($conn, "INSERT INTO clob_stream (id,my_clob) VALUES
(1, ?)");
   $variable = "THIS IS A CLOB TEST. THIS IS A CLOB TEST.";
   db2_bind_param($stmt, 1, "variable", DB2_PARAM_IN);
   db2_execute($stmt);

   $sql = "SELECT id,my_clob FROM clob_stream";
   $result = db2_prepare($conn, $sql);
   db2_execute($result);
   db2_fetch_row($result);
   $i = 0;
   /* Read LOB data */
   while ($data = db2_lob_read($result, 2, 6)) {
       echo "Loop $i: $data\n";
       $i = $i + 1;
   }

   $drop = 'DROP TABLE blob_stream';
   $result = @db2_exec( $conn, $drop );

   $create = 'CREATE TABLE blob_stream (id INTEGER, my_blob CLOB)';
   $result = db2_exec( $conn, $create );

   $variable = "";
   $stmt = db2_prepare($conn, "INSERT INTO blob_stream (id,my_blob) VALUES
(1, ?)");
   $variable = "THIS IS A BLOB TEST. THIS IS A BLOB TEST.";
   db2_bind_param($stmt, 1, "variable", DB2_PARAM_IN);
   db2_execute($stmt);

   $sql = "SELECT id,my_blob FROM blob_stream";
   $result = db2_prepare($conn, $sql);
   db2_execute($result);
   db2_fetch_row($result);
   $i = 0;
   /* Read LOB data */
   while ($data = db2_lob_read($result, 2, 6)) {
       echo "Loop $i: $data\n";
       $i = $i + 1;
   }
} else {
   echo 'no connection: ' . db2_conn_errormsg();
}

?>
```

The above example will output:

```
Loop 0: THIS I
Loop 1: S A CL
Loop 2: OB TES
Loop 3: T. THI
Loop 4: S IS A
Loop 5:  CLOB
Loop 6: TEST.
```

```
Loop 0: THIS I
Loop 1: S A BL
Loop 2: OB TES
Loop 3: T. THI
Loop 4: S IS A
Loop 5:  BLOB
Loop 6: TEST.
```

See Also

- db2_bind_param()
- db2_exec()
- db2_execute()
- db2_fetch_row()
- db2_prepare()
- db2_result()

db2_next_result

db2_next_result -- Requests the next result set from a stored procedure

Description

resource **db2_next_result** (resource `$stmt`)

A stored procedure can return zero or more result sets. While you handle the first result set in exactly the same way you would handle the results returned by a simple SELECT statement, to fetch the second and subsequent result sets from a stored procedure you must call the db2_next_result() function and return the result to a uniquely named PHP variable.

Parameters

`stmt`
A prepared statement returned from db2_exec() or db2_execute().

Return Values

Returns a new statement resource containing the next result set if the stored procedure returned another result set. Returns **FALSE** if the stored procedure did not return another result set.

Examples

Example #36 - Calling a stored procedure that returns multiple result sets

In the following example, we call a stored procedure that returns three result sets. The first result set is fetched directly from the same statement resource on which we invoked the CALL statement, while the second and third result sets are fetched from statement resources returned from our calls to the db2_next_result() function.

```
<?php
$conn = db2_connect($database, $user, $password);

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

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

 print "\nFetching second result set\n";
```

```
 $res = db2_next_result($stmt);
 if ($res) {
   while ($row = db2_fetch_array($res)) {
     var_dump($row);
   }
 }

 print "\nFetching third result set\n";
 $res2 = db2_next_result($stmt);
 if ($res2) {
   while ($row = db2_fetch_array($res2)) {
     var_dump($row);
   }
 }

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

The above example will output:

```
Fetching first result set
array(2) {
 [0]=>
 string(16) "Bubbles         "
 [1]=>
 int(3)
}
array(2) {
 [0]=>
 string(16) "Gizmo           "
 [1]=>
 int(4)
}

Fetching second result set
array(4) {
 [0]=>
 string(16) "Sweater         "
 [1]=>
 int(6)
 [2]=>
 string(5) "llama"
 [3]=>
 string(6) "150.00"
}
array(4) {
 [0]=>
 string(16) "Smarty          "
 [1]=>
 int(2)
 [2]=>
 string(5) "horse"
 [3]=>
 string(6) "350.00"
}

Fetching third result set
array(1) {
 [0]=>
```

```
 string(16) "Bubbles         "
}
array(1) {
 [0]=>
 string(16) "Gizmo           "
}
```

db2_num_fields

db2_num_fields -- Returns the number of fields contained in a result set

Description

int **db2_num_fields** (resource `$stmt`)

Returns the number of fields contained in a result set. This is most useful for handling the result sets returned by dynamically generated queries, or for result sets returned by stored procedures, where your application cannot otherwise know how to retrieve and use the results.

Parameters

`stmt`
A valid statement resource containing a result set.

Return Values

Returns an integer value representing the number of fields in the result set associated with the specified statement resource. Returns **FALSE** if the statement resource is not a valid input value.

Examples

Example #37 - Retrieving the number of fields in a result set

The following example demonstrates how to retrieve the number of fields returned in a result set.

```
<?php

$sql = "SELECT id, name, breed, weight FROM animals ORDER BY breed";
$stmt = db2_prepare($conn, $sql);
db2_execute($stmt, $sql);
$columns = db2_num_fields($stmt);

echo "There are {$columns} columns in the result set.";
?>
```

The above example will output:

```
There are 4 columns in the result set.
```

See Also

- db2_execute()
- db2_field_display_size()
- db2_field_name()
- db2_field_num()
- db2_field_precision()
- db2_field_scale()
- db2_field_type()
- db2_field_width()

db2_num_rows

db2_num_rows -- Returns the number of rows affected by an SQL statement

Description

int **db2_num_rows** (resource `$stmt`)

Returns the number of rows deleted, inserted, or updated by an SQL statement.

To determine the number of rows that will be returned by a SELECT statement, issue SELECT COUNT(*) with the same predicates as your intended SELECT statement and retrieve the value.

If your application logic checks the number of rows returned by a SELECT statement and branches if the number of rows is 0, consider modifying your application to attempt to return the first row with one of db2_fetch_assoc(), db2_fetch_both(), db2_fetch_array(), or db2_fetch_row(), and branch if the fetch function returns **FALSE**.

Note
If you issue a SELECT statement using a scrollable cursor, db2_num_rows() returns the number of rows returned by the SELECT statement. However, the overhead associated with scrollable cursors significantly degrades the performance of your application, so if this is the only reason you are considering using scrollable cursors, you should use a forward-only cursor and either call SELECT COUNT(*) or rely on the boolean return value of the fetch functions to achieve the equivalent functionality with much better performance.

Parameters

`stmt`
A valid *stmt* resource containing a result set.

Return Values

Returns the number of rows affected by the last SQL statement issued by the specified statement handle.

db2_pconnect

db2_pconnect -- Returns a persistent connection to a database

Description

resource **db2_pconnect** (string `$database`, string `$username`, string `$password` [, array `$options`])

Returns a persistent connection to an IBM DB2 Universal Database, IBM Cloudscape, or Apache Derby database. For more information on persistent connections, refer to Persistent Database Connections.

Calling db2_close() on a persistent connection always returns **TRUE**, but the underlying DB2 client connection remains open and waiting to serve the next matching db2_pconnect() request.

Parameters

`database`
The database alias in the DB2 client catalog.

`username`
The username with which you are connecting to the database.

`password`
The password with which you are connecting to the database.

`options`
An associative array of connection options that affect the behavior of the connection, where valid array keys include:

`autocommit`
Passing the *DB2_AUTOCOMMIT_ON* value turns autocommit on for this connection handle. Passing the *DB2_AUTOCOMMIT_OFF* value turns autocommit off for this connection handle.

`DB2_ATTR_CASE`
Passing the *DB2_CASE_NATURAL* value specifies that column names are returned in natural case. Passing the *DB2_CASE_LOWER* value specifies that column names are returned in lower case. Passing the *DB2_CASE_UPPER* value specifies that column names are returned in upper case.

`CURSOR`
Passing the *DB2_FORWARD_ONLY* value specifies a forward-only cursor for a statement resource. This is the default cursor type and is supported on all database servers. Passing the *DB2_SCROLLABLE* value specifies a scrollable cursor for a statement resource. This mode enables random access to rows in a result set, but currently is supported only by IBM DB2 Universal Database.

Return Values

Returns a connection handle resource if the connection attempt is successful. db2_pconnect() tries to reuse an existing connection resource that exactly matches the *database*, *username*, and *password* parameters. If the connection attempt fails, db2_pconnect() returns **FALSE**.

Examples

Example #38 - A db2_pconnect() example

In the following example, the first call to db2_pconnect() returns a new persistent connection resource. The second call to db2_pconnect() returns a persistent connection resource that simply reuses the first persistent connection resource.

```
<?php
$database = 'SAMPLE';
$user = 'db2inst1';
$password = 'ibmdb2';

$pconn = db2_pconnect($database, $user, $password);

if ($pconn) {
   echo "Persistent connection succeeded.";
}
else {
   echo "Persistent connection failed.";
}

$pconn2 = db2_pconnect($database, $user, $password);
if ($pconn) {
   echo "Second persistent connection succeeded.";
}
else {
   echo "Second persistent connection failed.";
}
?>
```

The above example will output:

```
Persistent connection succeeded.
Second persistent connection succeeded.
```

See Also

- db2_connect()

db2_prepare

db2_prepare -- Prepares an SQL statement to be executed

Description

resource **db2_prepare** (resource `$connection`, string `$statement` [, array `$options`])

db2_prepare() creates a prepared SQL statement which can include 0 or more parameter markers (*?* characters) representing parameters for input, output, or input/output. You can pass parameters to the prepared statement using db2_bind_param(), or for input values only, as an array passed to db2_execute().

There are three main advantages to using prepared statements in your application:

- *Performance*: when you prepare a statement, the database server creates an optimized access plan for retrieving data with that statement. Subsequently issuing the prepared statement with db2_execute() enables the statements to reuse that access plan and avoids the overhead of dynamically creating a new access plan for every statement you issue.
- *Security*: when you prepare a statement, you can include parameter markers for input values. When you execute a prepared statement with input values for placeholders, the database server checks each input value to ensure that the type matches the column definition or parameter definition.
- *Advanced functionality*: Parameter markers not only enable you to pass input values to prepared SQL statements, they also enable you to retrieve OUT and INOUT parameters from stored procedures using db2_bind_param().

Parameters

`connection`
: A valid database connection resource variable as returned from db2_connect() or db2_pconnect().

`statement`
: An SQL statement, optionally containing one or more parameter markers..

`options`
: An associative array containing statement options. You can use this parameter to request a scrollable cursor on database servers that support this functionality.

  `cursor`
  : Passing the *DB2_FORWARD_ONLY* value requests a forward-only cursor for this SQL statement. This is the default type of cursor, and it is supported by all database servers. It is also much faster than a scrollable cursor. Passing the *DB2_SCROLLABLE* value requests a scrollable cursor for this SQL statement.

This type of cursor enables you to fetch rows non-sequentially from the database server. However, it is only supported by DB2 servers, and is much slower than forward-only cursors.

Return Values

Returns a statement resource if the SQL statement was successfully parsed and prepared by the database server. Returns **FALSE** if the database server returned an error. You can determine which error was returned by calling db2_stmt_error() or db2_stmt_errormsg().

Examples

Example #39 - Preparing and executing an SQL statement with parameter markers

The following example prepares an INSERT statement that accepts four parameter markers, then iterates over an array of arrays containing the input values to be passed to db2_execute().

```
<?php
$animals = array(
   array(0, 'cat', 'Pook', 3.2),
   array(1, 'dog', 'Peaches', 12.3),
   array(2, 'horse', 'Smarty', 350.0),
);

$insert = 'INSERT INTO animals (id, breed, name, weight)
   VALUES (?, ?, ?, ?)';
$stmt = db2_prepare($conn, $insert);
if ($stmt) {
   foreach ($animals as $animal) {
       $result = db2_execute($stmt, $animal);
   }
}
?>
```

See Also

- db2_bind_param()
- db2_execute()
- db2_stmt_error()
- db2_stmt_errormsg()

db2_primary_keys

db2_primary_keys -- Returns a result set listing primary keys for a table

Description

resource **db2_primary_keys** (resource `$connection`, string `$qualifier`, string `$schema`, string `$table-name`)

Returns a result set listing the primary keys for a table.

Parameters

`connection`
A valid connection to an IBM DB2, Cloudscape, or Apache Derby database.

`qualifier`
A qualifier for DB2 databases running on OS/390 or z/OS servers. For other databases, pass **NULL** or an empty string.

`schema`
The schema which contains the tables. If *`schema`* is **NULL**, db2_primary_keys() matches the schema for the current connection.

`table-name`
The name of the table.

Return Values

Returns a statement resource with a result set containing rows describing the primary keys for the specified table. The result set is composed of the following columns:

Column name	Description
TABLE_CAT	Name of the catalog for the table containing the primary key. The value is NULL if this table does not have catalogs.
TABLE_SCHEM	Name of the schema for the table containing the primary key.
TABLE_NAME	Name of the table containing the primary key.
COLUMN_NAME	Name of the column containing the primary key.

KEY_SEQ	1-indexed position of the column in the key.
PK_NAME	The name of the primary key.

See Also

- db2_column_privileges()
- db2_columns()
- db2_foreign_keys()
- db2_procedure_columns()
- db2_procedures()
- db2_special_columns()
- db2_statistics()
- db2_table_privileges()
- db2_tables()

db2_procedure_columns

db2_procedure_columns -- Returns a result set listing stored procedure parameters

Description

resource **db2_procedure_columns** (resource `$connection`, string `$qualifier`, string `$schema`, string `$procedure`, string `$parameter`)

Returns a result set listing the parameters for one or more stored procedures.

Parameters

`connection`
A valid connection to an IBM DB2, Cloudscape, or Apache Derby database.

`qualifier`
A qualifier for DB2 databases running on OS/390 or z/OS servers. For other databases, pass **NULL** or an empty string.

`schema`
The schema which contains the procedures. This parameter accepts a search pattern containing _ and % as wildcards.

`procedure`
The name of the procedure. This parameter accepts a search pattern containing _ and % as wildcards.

`parameter`
The name of the parameter. This parameter accepts a search pattern containing _ and % as wildcards. If this parameter is **NULL**, all parameters for the specified stored procedures are returned.

Return Values

Returns a statement resource with a result set containing rows describing the parameters for the stored procedures matching the specified parameters. The rows are composed of the following columns:

Column name	Description
PROCEDURE_CAT	The catalog that contains the procedure. The value is **NULL** if this table does not have catalogs.
PROCEDURE_SCHEM	Name of the schema that contains the stored procedure.

PROCEDURE_NAME	Name of the procedure.
COLUMN_NAME	Name of the parameter.

COLUMN_TYPE

An integer value representing the type of the parameter:

Return value	**Parameter type**
1 (SQL_PARAM_INPUT)	Input (IN) parameter.
2 (SQL_PARAM_INPUT_OUTPUT)	Input/output (INOUT) parameter.
3 (SQL_PARAM_OUTPUT)	Output (OUT) parameter.

DATA_TYPE	The SQL data type for the parameter represented as an integer value.
TYPE_NAME	A string representing the data type for the parameter.
COLUMN_SIZE	An integer value representing the size of the parameter.
BUFFER_LENGTH	Maximum number of bytes necessary to store data for this parameter.
DECIMAL_DIGITS	The scale of the parameter, or **NULL** where scale is not applicable.
NUM_PREC_RADIX	An integer value of either *10* (representing an exact numeric data type), *2* (representing an approximate numeric data type), or **NULL** (representing a data type for which radix is not applicable).
NULLABLE	An integer value representing whether the parameter is nullable or not.
REMARKS	Description of the parameter.
COLUMN_DEF	Default value for the parameter.
SQL_DATA_TYPE	An integer value representing the size of the parameter.
SQL_DATETIME_SUB	Returns an integer value representing a datetime subtype code, or **NULL** for SQL

	data types to which this does not apply.
CHAR_OCTET_LENGTH	Maximum length in octets for a character data type parameter, which matches COLUMN_SIZE for single-byte character set data, or **NULL** for non-character data types.
ORDINAL_POSITION	The 1-indexed position of the parameter in the CALL statement.
IS_NULLABLE	A string value where 'YES' means that the parameter accepts or returns **NULL** values and 'NO' means that the parameter does not accept or return **NULL** values.

See Also

- db2_column_privileges()
- db2_columns()
- db2_foreign_keys()
- db2_primary_keys()
- db2_procedures()
- db2_special_columns()
- db2_statistics()
- db2_table_privileges()
- db2_tables()

db2_procedures

db2_procedures -- Returns a result set listing the stored procedures registered in a database

Description

resource **db2_procedures** (resource `$connection`, string `$qualifier`, string `$schema`, string `$procedure`)

Returns a result set listing the stored procedures registered in a database.

Parameters

`connection`
A valid connection to an IBM DB2, Cloudscape, or Apache Derby database.

`qualifier`
A qualifier for DB2 databases running on OS/390 or z/OS servers. For other databases, pass **NULL** or an empty string.

`schema`
The schema which contains the procedures. This parameter accepts a search pattern containing _ and % as wildcards.

`procedure`
The name of the procedure. This parameter accepts a search pattern containing _ and % as wildcards.

Return Values

Returns a statement resource with a result set containing rows describing the stored procedures matching the specified parameters. The rows are composed of the following columns:

Column name	Description
PROCEDURE_CAT	The catalog that contains the procedure. The value is **NULL** if this table does not have catalogs.
PROCEDURE_SCHEM	Name of the schema that contains the stored procedure.
PROCEDURE_NAME	Name of the procedure.
NUM_INPUT_PARAMS	Number of input (IN) parameters for the

	stored procedure.
NUM_OUTPUT_PARAMS	Number of output (OUT) parameters for the stored procedure.
NUM_RESULT_SETS	Number of result sets returned by the stored procedure.
REMARKS	Any comments about the stored procedure.
PROCEDURE_TYPE	Always returns *1*, indicating that the stored procedure does not return a return value.

See Also

- db2_column_privileges()
- db2_columns()
- db2_foreign_keys()
- db2_primary_keys()
- db2_procedure_columns()
- db2_special_columns()
- db2_statistics()
- db2_table_privileges()
- db2_tables()

db2_result

db2_result -- Returns a single column from a row in the result set

Description

mixed **db2_result** (resource `$stmt`, mixed `$column`)

Use db2_result() to return the value of a specified column in the current row of a result set. You must call db2_fetch_row() before calling db2_result() to set the location of the result set pointer.

Parameters

`stmt`
A valid *stmt* resource.

`column`
Either an integer mapping to the 0-indexed field in the result set, or a string matching the name of the column.

Return Values

Returns the value of the requested field if the field exists in the result set. Returns NULL if the field does not exist, and issues a warning.

Examples

Example #40 - A db2_result() example

The following example demonstrates how to iterate through a result set with db2_fetch_row() and retrieve columns from the result set with db2_result().

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

The above example will output:

```
cat Pook
gold fish Bubbles
budgerigar Gizmo
goat Rickety Ride
```

See Also

- db2_fetch_array()
- db2_fetch_assoc()
- db2_fetch_both()
- db2_fetch_object()
- db2_fetch_row()

db2_rollback

db2_rollback -- Rolls back a transaction

Description

bool **db2_rollback** (resource `$connection`)

Rolls back an in-progress transaction on the specified connection resource and begins a new transaction. PHP applications normally default to AUTOCOMMIT mode, so db2_rollback() normally has no effect unless AUTOCOMMIT has been turned off for the connection resource.

Note
If the specified connection resource is a persistent connection, all transactions in progress for all applications using that persistent connection will be rolled back. For this reason, persistent connections are not recommended for use in applications that require transactions.

Parameters

`connection`
A valid database connection resource variable as returned from db2_connect() or db2_pconnect().

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #41 - Rolling back a DELETE statement

In the following example, we count the number of rows in a table, turn off AUTOCOMMIT mode on a database connection, delete all of the rows in the table and return the count of *0* to prove that the rows have been removed. We then issue db2_rollback() and return the updated count of rows in the table to show that the number is the same as before we issued the DELETE statement. The return to the original state of the table demonstrates that the roll back of the transaction succeeded.
 	
`<?php`
`$conn = db2_connect($database, $user, $password);`

```
if ($conn) {
   $stmt = db2_exec($conn, "SELECT count(*) FROM animals");
   $res = db2_fetch_array( $stmt );
   echo $res[0] . "\n";

   // Turn AUTOCOMMIT off
   db2_autocommit($conn, DB2_AUTOCOMMIT_OFF);

   // Delete all rows from ANIMALS
   db2_exec($conn, "DELETE FROM animals");

   $stmt = db2_exec($conn, "SELECT count(*) FROM animals");
   $res = db2_fetch_array( $stmt );
   echo $res[0] . "\n";

   // Roll back the DELETE statement
   db2_rollback( $conn );

   $stmt = db2_exec( $conn, "SELECT count(*) FROM animals" );
   $res = db2_fetch_array( $stmt );
   echo $res[0] . "\n";
   db2_close($conn);
}
?>
```

The above example will output:

```
7
0
7
```

See Also

- db2_autocommit()
- db2_commit()

db2_server_info

db2_server_info -- Returns an object with properties that describe the DB2 database server

Description

object **db2_server_info** (resource `$connection`)

This function returns an object with read-only properties that return information about the IBM DB2, Cloudscape, or Apache Derby database server. The following table lists the database server properties:

Database server properties

Property name	Return type	Description
DBMS_NAME	string	The name of the database server to which you are connected. For DB2 servers this is a combination of *DB2* followed by the operating system on which the database server is running.
DBMS_VER	string	The version of the database server, in the form of a string "MM.mm.uuuu" where *MM* is the major version, *mm* is the minor version, and *uuuu* is the update. For example, "08.02.0001" represents major version 8, minor version 2, update 1.
DB_CODEPAGE	int	The code page of the database to which you are connected.
DB_NAME	string	The name of the database to which you are connected.
DFT_ISOLATION	string	The default transaction isolation level supported by the server: UR Uncommitted read: changes are immediately

		visible by all concurrent transactions. CS Cursor stability: a row read by one transaction can be altered and committed by a second concurrent transaction. RS Read stability: a transaction can add or remove rows matching a search condition or a pending transaction. RR Repeatable read: data affected by pending transaction is not available to other transactions. NC No commit: any changes are visible at the end of a successful operation. Explicit commits and rollbacks are not allowed.
IDENTIFIER_QUOTE_CHAR	string	The character used to delimit an identifier.
INST_NAME	string	The instance on the database server that contains the database.
ISOLATION_OPTION	array	An array of the isolation options supported by the database server. The isolation options are described in the DFT_ISOLATION property.
KEYWORDS	array	An array of the keywords reserved by the database server.
LIKE_ESCAPE_CLAUSE	bool	**TRUE** if the database server

		supports the use of % and _ wildcard characters. **FALSE** if the database server does not support these wildcard characters.
MAX_COL_NAME_LEN	int	Maximum length of a column name supported by the database server, expressed in bytes.
MAX_IDENTIFIER_LEN	int	Maximum length of an SQL identifier supported by the database server, expressed in characters.
MAX_INDEX_SIZE	int	Maximum size of columns combined in an index supported by the database server, expressed in bytes.
MAX_PROC_NAME_LEN	int	Maximum length of a procedure name supported by the database server, expressed in bytes.
MAX_ROW_SIZE	int	Maximum length of a row in a base table supported by the database server, expressed in bytes.
MAX_SCHEMA_NAME_LEN	int	Maximum length of a schema name supported by the database server, expressed in bytes.
MAX_STATEMENT_LEN	int	Maximum length of an SQL statement supported by the database server, expressed in bytes.
MAX_TABLE_NAME_LEN	int	Maximum length of a table name supported by the database server, expressed in bytes.
NON_NULLABLE_COLUMNS	bool	**TRUE** if the database server supports columns that can be defined as NOT NULL, **FALSE** if the database server does not support columns defined as NOT

		NULL.
PROCEDURES	bool	**TRUE** if the database server supports the use of the CALL statement to call stored procedures, **FALSE** if the database server does not support the CALL statement.
SPECIAL_CHARS	string	A string containing all of the characters other than a-Z, 0-9, and underscore that can be used in an identifier name.
SQL_CONFORMANCE	string	The level of conformance to the ANSI/ISO SQL-92 specification offered by the database server: ENTRY Entry-level SQL-92 compliance. FIPS127 FIPS-127-2 transitional compliance. FULL Full level SQL-92 compliance. INTERMEDIATE Intermediate level SQL-92 compliance.

Parameters

connection
Specifies an active DB2 client connection.

Return Values

Returns an object on a successful call. Returns **FALSE** on failure.

Examples

Example #42 - A db2_server_info() example

To retrieve information about the server, you must pass a valid database connection resource to db2_server_info().

```
<?php

$conn = db2_connect('sample', 'db2inst1', 'ibmdb2');

$server = db2_server_info( $conn );

if ($server) {
   echo "DBMS_NAME: ";                 var_dump( $server->DBMS_NAME );
   echo "DBMS_VER: ";                  var_dump( $server->DBMS_VER );
   echo "DB_CODEPAGE: ";               var_dump( $server->DB_CODEPAGE );
   echo "DB_NAME: ";                   var_dump( $server->DB_NAME );
   echo "INST_NAME: ";                 var_dump( $server->INST_NAME );
   echo "SPECIAL_CHARS: ";             var_dump( $server->SPECIAL_CHARS );
   echo "KEYWORDS: ";                  var_dump( sizeof($server->KEYWORDS)
);
   echo "DFT_ISOLATION: ";             var_dump( $server->DFT_ISOLATION );
   echo "ISOLATION_OPTION: ";
   $il = '';
   foreach( $server->ISOLATION_OPTION as $opt )
   {
      $il .= $opt." ";
   }
   var_dump( $il );
   echo "SQL_CONFORMANCE: ";           var_dump( $server->SQL_CONFORMANCE );
   echo "PROCEDURES: ";                var_dump( $server->PROCEDURES );
   echo "IDENTIFIER_QUOTE_CHAR: ";     var_dump(
$server->IDENTIFIER_QUOTE_CHAR );
   echo "LIKE_ESCAPE_CLAUSE: ";        var_dump( $server->LIKE_ESCAPE_CLAUSE
);
   echo "MAX_COL_NAME_LEN: ";          var_dump( $server->MAX_COL_NAME_LEN
);
   echo "MAX_ROW_SIZE: ";              var_dump( $server->MAX_ROW_SIZE );
   echo "MAX_IDENTIFIER_LEN: ";        var_dump( $server->MAX_IDENTIFIER_LEN
);
   echo "MAX_INDEX_SIZE: ";            var_dump( $server->MAX_INDEX_SIZE );
   echo "MAX_PROC_NAME_LEN: ";         var_dump( $server->MAX_PROC_NAME_LEN
);
   echo "MAX_SCHEMA_NAME_LEN: ";       var_dump(
$server->MAX_SCHEMA_NAME_LEN );
   echo "MAX_STATEMENT_LEN: ";         var_dump( $server->MAX_STATEMENT_LEN
);
   echo "MAX_TABLE_NAME_LEN: ";        var_dump( $server->MAX_TABLE_NAME_LEN
);
   echo "NON_NULLABLE_COLUMNS: ";      var_dump(
$server->NON_NULLABLE_COLUMNS );

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

The above example will output:

```
DBMS_NAME: string(9) "DB2/LINUX"
DBMS_VER: string(10) "08.02.0000"
DB_CODEPAGE: int(1208)
DB_NAME: string(6) "SAMPLE"
INST_NAME: string(8) "db2inst1"
SPECIAL_CHARS: string(2) "@#"
KEYWORDS: int(179)
DFT_ISOLATION: string(2) "CS"
ISOLATION_OPTION: string(12) "UR CS RS RR "
SQL_CONFORMANCE: string(7) "FIPS127"
PROCEDURES: bool(true)
IDENTIFIER_QUOTE_CHAR: string(1) """
LIKE_ESCAPE_CLAUSE: bool(true)
MAX_COL_NAME_LEN: int(30)
MAX_ROW_SIZE: int(32677)
MAX_IDENTIFIER_LEN: int(18)
MAX_INDEX_SIZE: int(1024)
MAX_PROC_NAME_LEN: int(128)
MAX_SCHEMA_NAME_LEN: int(30)
MAX_STATEMENT_LEN: int(2097152)
MAX_TABLE_NAME_LEN: int(128)
NON_NULLABLE_COLUMNS: bool(true)
```

See Also

- db2_client_info()

db2_set_option

db2_set_option -- Set options for connection or statement resources

Description

bool **db2_set_option** (resource `$resource`, array `$options`, int `$type`)

Sets options for a statement resource or a connection resource. You cannot set options for result set resources.

Parameters

`resource`
A valid statement resource as returned from db2_prepare() or a valid connection resource as returned from db2_connect() or db2_pconnect().

`options`
An associative array containing valid statement or connection options. This parameter can be used to change autocommit values, cursor types (scrollable or forward), and to specify the case of the column names (lower, upper, or natural) that will appear in a result set.

`autocommit`
Passing *DB2_AUTOCOMMIT_ON* turns autocommit on for the specified connection resource. Passing *DB2_AUTOCOMMIT_OFF* turns autocommit off for the specified connection resource.

`cursor`
Passing *DB2_FORWARD_ONLY* specifies a forward-only cursor for a statement resource. This is the default cursor type, and is supported by all database servers. Passing *DB2_SCROLLABLE* specifies a scrollable cursor for a statement resource. Scrollable cursors enable result set rows to be accessed in non-sequential order, but are only supported by IBM DB2 Universal Database databases.

`binmode`
Passing *DB2_BINARY* specifies that binary data will be returned as is. This is the default mode. This is the equivalent of setting *ibm_db2.binmode=1* in *php.ini*. Passing *DB2_CONVERT* specifies that binary data will be converted to hexadecimal encoding, and will be returned as such. This is the equivalent of setting *ibm_db2.binmode=2* in *php.ini*. Passing *DB2_PASSTHRU* specifies that binary data will be converted to **NULL**. This is the equivalent of setting *ibm_db2.binmode=3* in *php.ini*.

`db2_attr_case`
Passing *DB2_CASE_LOWER* specifies that column names of the result set are returned in lower case. Passing *DB2_CASE_UPPER* specifies that column names of the result set are returned in upper case. Passing *DB2_CASE_NATURAL*

specifies that column names of the result set are returned in natural case.

deferred_prepare
Passing *DB2_DEFERRED_PREPARE_ON* turns deferred prepare on for the specified statement resource. Passing *DB2_DEFERRED_PREPARE_OFF* turns deferred prepare off for the specified statement resource.

The following new i5/OS options are available as of ibm_db2 version 1.5.1.

Note

Note

Prior versions of ibm_db2 do not support these new i5 options.

i5_fetch_only
DB2_I5_FETCH_ON - Cursors are read-only and cannot be used for positioned updates or deletes. This is the default unless *SQL_ATTR_FOR_FETCH_ONLY* environment has been set to *SQL_FALSE*. *DB2_I5_FETCH_OFF* - Cursors can be used for positioned updates and deletes.

The following new options are available as of ibm_db2 version 1.6.0. They provide useful tracking information that can be accessed during execution with db2_get_option().

Note

Note

Prior versions of ibm_db2 do not support these new options.

When the value in each option is being set, some servers might not handle the entire length provided and might truncate the value.

To ensure that the data specified in each option is converted correctly when transmitted to a host system, use only the characters A through Z, 0 through 9, and the underscore (_) or period (.).

userid
SQL_ATTR_INFO_USERID - A pointer to a null-terminated character string used to identify the client user ID sent to the host database server when using DB2 Connect.

Note

Note

DB2 for z/OS and OS/390 servers support up to a length of 16 characters. This user-id is not to be confused with the authentication user-id, it is for identification purposes only and is not used for any authorization.

acctstr

SQL_ATTR_INFO_ACCTSTR - A pointer to a null-terminated character string used to identify the client accounting string sent to the host database server when using DB2 Connect.

Note
Note DB2 for z/OS and OS/390 servers support up to a length of 200 characters.

applname

SQL_ATTR_INFO_APPLNAME - A pointer to a null-terminated character string used to identify the client application name sent to the host database server when using DB2 Connect.

Note
Note DB2 for z/OS and OS/390 servers support up to a length of 32 characters.

wrkstnname

SQL_ATTR_INFO_WRKSTNNAME - A pointer to a null-terminated character string used to identify the client workstation name sent to the host database server when using DB2 Connect.

Note
Note DB2 for z/OS and OS/390 servers support up to a length of 18 characters.

type

An integer value that specifies the type of resource that was passed into the function. The type of resource and this value must correspond.

Passing *1* as the value specifies that a connection resource has been passed into the function. Passing any integer not equal to *1* as the value specifies that a statement resource has been passed into the function.

The following table specifies which options are compatible with the available resource types:

Resource-Parameter Matrix

Key	**Value**	**Resource Type**		
		Connection	Statement	Result Set
autocommit	*DB2_AUTOCOMMIT_ON*	X	-	-
autocommit	*DB2_AUTOCOMMIT_OFF*	X	-	-
cursor	*DB2_SCROLLABLE*	-	X	-
cursor	*DB2_FORWARD_ONLY*	-	X	-
binmode	*DB2_BINARY*	X	X	-
binmode	*DB2_CONVERT*	X	X	-
binmode	*DB2_PASSTHRU*	X	X	-
db2_attr_case	*DB2_CASE_LOWER*	X	X	-
db2_attr_case	*DB2_CASE_UPPER*	X	X	-
db2_attr_case	*DB2_CASE_NATURAL*	X	X	-
deferred_prepare	*DB2_DEFERRED_PREPARE_ON*	-	X	-
deferred_prepare	*DB2_DEFERRED_PREPARE_OFF*	-	X	-
i5_fetch_only	*DB2_I5_FETCH_ON*	-	X	-
i5_fetch_only	*DB2_I5_FETCH_OFF*	-	X	-
userid	*SQL_ATTR_INF*	*X*	*X*	-

	O_USERID			
acctstr	*SQL_ATTR_INFO_ACCTSTR*	X	X	-
applname	*SQL_ATTR_INFO_APPLNAME*	X	X	-
wrkstnname	*SQL_ATTR_INFO_WRKSTNNAME*	X	X	-

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #43 - Setting one parameter with a connection resource

```
<?php
/* Database Connection Parameters */
$database = 'SAMPLE';
$hostname = 'localhost';
$port = 50000;
$protocol = 'TCPIP';
$username = 'db2inst1';
$password = 'ibmdb2';

/* Connection String */
$conn_string = "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=$database;";
$conn_string .= "HOSTNAME=$hostname;PORT=$port;PROTOCOL=$protocol;";
$conn_string .= "UID=$username;PWD=$password;";

/* Obtain Connection Resource */
$conn = db2_connect($conn_string, '', '');

/* Create the associative options array with valid key-value pairs */
$options = array('autocommit' => DB2_AUTOCOMMIT_ON);

/* Call the function using the correct resource, options array, and type
values */
$result = db2_set_option($conn, $options, 1);

/* Check if all options could be set correctly */
if($result)
{
 echo 'Options Set Successfully';
}
```

```
else
{
 echo 'Could Not Set Options';
}
?>
```

The above example will output:

```
Options Set Successfully
```

Example #44 - Setting multiple parameters with a connection resource

```
<?php
/* Database Connection Parameters */
$database = 'SAMPLE';
$hostname = 'localhost';
$port = 50000;
$protocol = 'TCPIP';
$username = 'db2inst1';
$password = 'ibmdb2';

/* Connection String */
$conn_string = "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=$database;";
$conn_string .= "HOSTNAME=$hostname;PORT=$port;PROTOCOL=$protocol;";
$conn_string .= "UID=$username;PWD=$password;";

/* Obtain Connection Resource */
$conn = db2_connect($conn_string, '', '');

/* Create the associative options array with valid key-value pairs */
$options = array('autocommit' => DB2_AUTOCOMMIT_OFF,
                 'binmode' => DB2_PASSTHRU,
             'db2_attr_case' => DB2_CASE_UPPER,
                  'cursor' => DB2_SCROLLABLE);

/* Call the function using the correct resource, options array, and type
values */
$result = db2_set_option($conn, $options, 1);

/* Check if all options could be set correctly */
if($result)
{
 echo 'Options Set Successfully';
}
else
{
 echo 'Could Not Set Options';
}
?>
```

The above example will output:

```
Options Set Successfully
```

Example #45 - Setting multiple parameters with an invalid key

```
<?php
/* Database Connection Parameters */
$database = 'SAMPLE';
$hostname = 'localhost';
$port = 50000;
$protocol = 'TCPIP';
$username = 'db2inst1';
$password = 'ibmdb2';

/* Connection String */
$conn_string = "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=$database;";
$conn_string .= "HOSTNAME=$hostname;PORT=$port;PROTOCOL=$protocol;";
$conn_string .= "UID=$username;PWD=$password;";

/* Obtain Connection Resource */
$conn = db2_connect($conn_string, '', '');

/* Create the associative options array with valid key-value pairs */
$options = array('autocommit' => DB2_AUTOCOMMIT_OFF,
            'MY_INVALID_KEY' => DB2_PASSTHRU,
             'db2_attr_case' => DB2_CASE_UPPER,
                    'cursor' => DB2_SCROLLABLE);

/* Call the function using the correct resource, options array, and type
values */
$result = db2_set_option($conn, $options, 1);

/* Check if all options could be set correctly */
if($result)
{
 echo 'Options Set Successfully';
}
else
{
 echo 'Could Not Set Options';
}
?>
```

The above example will output:

```
Could Not Set Options
```

Example #46 - Setting multiple parameters with an invalid value

```
<?php
/* Database Connection Parameters */
$database = 'SAMPLE';
$hostname = 'localhost';
$port = 50000;
```

```
$protocol = 'TCPIP';
$username = 'db2inst1';
$password = 'ibmdb2';

/* Connection String */
$conn_string = "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=$database;";
$conn_string .= "HOSTNAME=$hostname;PORT=$port;PROTOCOL=$protocol;";
$conn_string .= "UID=$username;PWD=$password;";

/* Obtain Connection Resource */
$conn = db2_connect($conn_string, '', '');

/* Create the associative options array with valid key-value pairs */
$options = array('autocommit' => DB2_AUTOCOMMIT_OFF,
                   'binmode' => 'INVALID_VALUE',
              'db2_attr_case' => DB2_CASE_UPPER,
                    'cursor' => DB2_SCROLLABLE);

/* Call the function using the correct resource, options array, and type
values */
$result = db2_set_option($conn, $options, 1);

/* Check if all options could be set correctly */
if($result)
{
 echo 'Options Set Successfully';
}
else
{
 echo 'Could Not Set Options';
}
?>
```

The above example will output:

```
Could Not Set Options
```

Example #47 - Setting multiple parameters with a connection resource and the wrong type

```
<?php
/* Database Connection Parameters */
$database = 'SAMPLE';
$hostname = 'localhost';
$port = 50000;
$protocol = 'TCPIP';
$username = 'db2inst1';
$password = 'ibmdb2';

/* Connection String */
$conn_string = "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=$database;";
$conn_string .= "HOSTNAME=$hostname;PORT=$port;PROTOCOL=$protocol;";
$conn_string .= "UID=$username;PWD=$password;";

/* Obtain Connection Resource */
```

```
$conn = db2_connect($conn_string, '', '');

/* Create the associative options array with valid key-value pairs */
$options = array('autocommit' => DB2_AUTOCOMMIT_OFF,
                 'binmode' => DB2_PASSTHRU,
             'db2_attr_case' => DB2_CASE_UPPER,
                   'cursor' => DB2_SCROLLABLE);

/* Call the function using the correct resource, options array, and the
wrong type value */
$result = db2_set_option($conn, $options, 2);

/* Check if all options could be set correctly */
if($result)
{
 echo 'Options Set Successfully';
}
else
{
 echo 'Could Not Set Options';
}
?>
```

The above example will output:

```
Could Not Set Options
```

Example #48 - Setting multiple parameters with the wrong resource

```
<?php
/* Database Connection Parameters */
$database = 'SAMPLE';
$hostname = 'localhost';
$port = 50000;
$protocol = 'TCPIP';
$username = 'db2inst1';
$password = 'ibmdb2';

/* Connection String */
$conn_string = "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=$database;";
$conn_string .= "HOSTNAME=$hostname;PORT=$port;PROTOCOL=$protocol;";
$conn_string .= "UID=$username;PWD=$password;";

/* Obtain Connection Resource */
$conn = db2_connect($conn_string, '', '');

/* Create the associative options array with valid key-value pairs */
$options = array('autocommit' => DB2_AUTOCOMMIT_OFF,
                 'binmode' => DB2_PASSTHRU,
             'db2_attr_case' => DB2_CASE_UPPER,
                   'cursor' => DB2_SCROLLABLE);

$stmt = db2_prepare($conn, 'SELECT * FROM EMPLOYEE');

/* Call the function using the wrong resource, and the correct options
```

```
array, and type values */
$result = db2_set_option($stmt, $options, 1);

/* Check if all options could be set correctly */
if($result)
{
 echo 'Options Set Successfully';
}
else
{
 echo 'Could Not Set Options';
}
?>
```

The above example will output:

```
Could Not Set Options
```

Example #49 - Putting it all together

```
<?php
/* Database Connection Parameters */
$database = 'SAMPLE';
$hostname = 'localhost';
$port = 50000;
$protocol = 'TCPIP';
$username = 'db2inst1';
$password = 'ibmdb2';

/* Connection String */
$conn_string = "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=$database;";
$conn_string .= "HOSTNAME=$hostname;PORT=$port;PROTOCOL=$protocol;";
$conn_string .= "UID=$username;PWD=$password;";

/* Obtain Connection Resource */
$conn = db2_connect($conn_string, '', '');

/* Create the associative options array with valid key-value pairs */
$options = array('db2_attr_case' => DB2_CASE_LOWER,
                       'cursor' => DB2_SCROLLABLE);

$stmt = db2_prepare($conn, 'SELECT * FROM EMPLOYEE WHERE EMPNO = ? OR EMPNO
= ?');

/* Call the function using the correct resource, options array, and type
values */
$option_result = db2_set_option($stmt, $options, 2);
$result = db2_execute($stmt, array('000130', '000140'));

/* Get Row 2 before Row 1 since Scrollable Cursor */
print_r(db2_fetch_assoc($stmt, 2));
print '<br /><br />';
print_r(db2_fetch_assoc($stmt, 1));

?>
```

The above example will output:

```
Array
(
   [empno] => 000140
   [firstnme] => HEATHER
   [midinit] => A
   [lastname] => NICHOLLS
   [workdept] => C01
   [phoneno] => 1793
   [hiredate] => 1976-12-15
   [job] => ANALYST
   [edlevel] => 18
   [sex] => F
   [birthdate] => 1946-01-19
   [salary] => 28420.00
   [bonus] => 600.00
   [comm] => 2274.00
)

Array
(
   [empno] => 000130
   [firstnme] => DELORES
   [midinit] => M
   [lastname] => QUINTANA
   [workdept] => C01
   [phoneno] => 4578
   [hiredate] => 1971-07-28
   [job] => ANALYST
   [edlevel] => 16
   [sex] => F
   [birthdate] => 1925-09-15
   [salary] => 23800.00
   [bonus] => 500.00
   [comm] => 1904.00
)
```

Example #50 - i5/OS cursors are read-only

```
<?php
 $conn = db2_connect("", "", "", array("i5_lib"=>"nobody"));
 $stmt = db2_prepare($conn, 'select * from names where first = ?');
 $name = "first2";
 db2_bind_param($stmt, 1, "name", DB2_PARAM_IN);
 $options = array("i5_fetch_only"=>DB2_I5_FETCH_ON);
 db2_set_option($stmt,$options,0);
 if (db2_execute($stmt)) {
   while ($row = db2_fetch_array($stmt)) {
     echo "{$row[0]} {$row[1]}";
   }
 }
?>
```

The above example will output:

```
first2 last2
```

See Also

- db2_connect()
- db2_pconnect()
- db2_exec()
- db2_prepare()
- db2_cursor_type()

db2_special_columns

db2_special_columns -- Returns a result set listing the unique row identifier columns for a table

Description

resource **db2_special_columns** (resource `$connection`, string `$qualifier`, string `$schema`, string `$table_name`, int `$scope`)

Returns a result set listing the unique row identifier columns for a table.

Parameters

`connection`
A valid connection to an IBM DB2, Cloudscape, or Apache Derby database.

`qualifier`
A qualifier for DB2 databases running on OS/390 or z/OS servers. For other databases, pass **NULL** or an empty string.

`schema`
The schema which contains the tables.

`table_name`
The name of the table.

`scope`
Integer value representing the minimum duration for which the unique row identifier is valid. This can be one of the following values:

Integer value	SQL constant	Description
0	SQL_SCOPE_CURROW	Row identifier is valid only while the cursor is positioned on the row.
1	SQL_SCOPE_TRANSACTION	Row identifier is valid for the duration of the transaction.
2	SQL_SCOPE_SESSION	Row identifier is valid for the duration of the connection.

Return Values

Returns a statement resource with a result set containing rows with unique row identifier information for a table. The rows are composed of the following columns:

Column name	Description

SCOPE

Integer value	SQL constant	Description
0	SQL_SCOPE_CURROW	Row identifier is valid only while the cursor is positioned on the row.
1	SQL_SCOPE_TRANSACTION	Row identifier is valid for the duration of the transaction.
2	SQL_SCOPE_SESSION	Row identifier is valid for the duration of the connection.

COLUMN_NAME	Name of the unique column.	
DATA_TYPE	SQL data type for the column.	
TYPE_NAME	Character string representation of the SQL data type for the column.	
COLUMN_SIZE	An integer value representing the size of the column.	
BUFFER_LENGTH	Maximum number of bytes necessary to store data from this column.	
DECIMAL_DIGITS	The scale of the column, or **NULL** where scale is not applicable.	
NUM_PREC_RADIX	An integer value of either *10* (representing an exact numeric data type), *2* (representing an approximate numeric data type), or **NULL** (representing a data type for which radix is not applicable).	

PSEUDO_COLUMN	Always returns 1.	

See Also

- db2_column_privileges()
- db2_columns()
- db2_foreign_keys()
- db2_primary_keys()
- db2_procedure_columns()
- db2_procedures()
- db2_statistics()
- db2_table_privileges()
- db2_tables()

db2_statistics

db2_statistics -- Returns a result set listing the index and statistics for a table

Description

resource **db2_statistics** (resource `$connection`, string `$qualifier`, string `$schema`, string `$table-name`, bool `$unique`)

Returns a result set listing the index and statistics for a table.

Parameters

`connection`
A valid connection to an IBM DB2, Cloudscape, or Apache Derby database.

`qualifier`
A qualifier for DB2 databases running on OS/390 or z/OS servers. For other databases, pass **NULL** or an empty string.

`schema`
The schema that contains the targeted table. If this parameter is **NULL**, the statistics and indexes are returned for the schema of the current user.

`table_name`
The name of the table.

`unique`
An integer value representing the type of index information to return.
`0`
Return only the information for unique indexes on the table.

`1`
Return the information for all indexes on the table.

Return Values

Returns a statement resource with a result set containing rows describing the statistics and indexes for the base tables matching the specified parameters. The rows are composed of the following columns:

Column name	Description
TABLE_CAT	The catalog that contains the table. The value is **NULL** if this table does not have catalogs.

TABLE_SCHEM	Name of the schema that contains the table.
TABLE_NAME	Name of the table.

NON_UNIQUE

An integer value representing whether the index prohibits unique values, or whether the row represents statistics on the table itself:

Return value	**Parameter type**
0 (SQL_FALSE)	The index allows duplicate values.
1 (SQL_TRUE)	The index values must be unique.
NULL	This row is statistics information for the table itself.

INDEX_QUALIFIER	A string value representing the qualifier that would have to be prepended to INDEX_NAME to fully qualify the index.
INDEX_NAME	A string representing the name of the index.

TYPE

An integer value representing the type of information contained in this row of the result set:

Return value	**Parameter type**
0 (SQL_TABLE_STAT)	The row contains statistics about the table itself.
1 (SQL_INDEX_CLUSTERED)	The row contains information about a clustered index.
2 (SQL_INDEX_HASH)	The row contains information about a hashed index.
3 (SQL_INDEX_OTHER)	The row contains information about a type of index that is neither clustered nor hashed.

ORDINAL_POSITION	The 1-indexed position of the column in the index. **NULL** if the row contains statistics

	information about the table itself.
COLUMN_NAME	The name of the column in the index. **NULL** if the row contains statistics information about the table itself.
ASC_OR_DESC	*A* if the column is sorted in ascending order, *D* if the column is sorted in descending order, **NULL** if the row contains statistics information about the table itself.
CARDINALITY	If the row contains information about an index, this column contains an integer value representing the number of unique values in the index. If the row contains information about the table itself, this column contains an integer value representing the number of rows in the table.
PAGES	If the row contains information about an index, this column contains an integer value representing the number of pages used to store the index. If the row contains information about the table itself, this column contains an integer value representing the number of pages used to store the table.
FILTER_CONDITION	Always returns **NULL**.

See Also

- db2_column_privileges()
- db2_columns()
- db2_foreign_keys()
- db2_primary_keys()
- db2_procedure_columns()
- db2_procedures()
- db2_special_columns()
- db2_table_privileges()

- db2_tables()

db2_stmt_error

db2_stmt_error -- Returns a string containing the SQLSTATE returned by an SQL statement

Description

string **db2_stmt_error** ([resource `$stmt`])

Returns a string containing the SQLSTATE value returned by an SQL statement.

If you do not pass a statement resource as an argument to db2_stmt_error(), the driver returns the SQLSTATE value associated with the last attempt to return a statement resource, for example, from db2_prepare() or db2_exec().

To learn what the SQLSTATE value means, you can issue the following command at a DB2 Command Line Processor prompt: **db2 '?** *`sqlstate-value`* **'**. You can also call db2_stmt_errormsg() to retrieve an explicit error message and the associated SQLCODE value.

Parameters

`stmt`
A valid statement resource.

Return Values

Returns a string containing an SQLSTATE value.

See Also

- db2_conn_error()
- db2_conn_errormsg()
- db2_stmt_errormsg()

db2_stmt_errormsg

db2_stmt_errormsg -- Returns a string containing the last SQL statement error message

Description

string **db2_stmt_errormsg** ([resource `$stmt`])

Returns a string containing the last SQL statement error message.

If you do not pass a statement resource as an argument to db2_stmt_errormsg(), the driver returns the error message associated with the last attempt to return a statement resource, for example, from db2_prepare() or db2_exec().

Parameters

`stmt`
: A valid statement resource.

Return Values

Returns a string containing the error message and SQLCODE value for the last error that occurred issuing an SQL statement.

See Also

- db2_conn_error()
- db2_conn_errormsg()
- db2_stmt_error()

db2_table_privileges

db2_table_privileges -- Returns a result set listing the tables and associated privileges in a database

Description

resource **db2_table_privileges** (resource `$connection` [, string `$qualifier` [, string `$schema` [, string `$table_name`]]])

Returns a result set listing the tables and associated privileges in a database.

Parameters

`connection`
A valid connection to an IBM DB2, Cloudscape, or Apache Derby database.

`qualifier`
A qualifier for DB2 databases running on OS/390 or z/OS servers. For other databases, pass **NULL** or an empty string.

`schema`
The schema which contains the tables. This parameter accepts a search pattern containing _ and % as wildcards.

`table_name`
The name of the table. This parameter accepts a search pattern containing _ and % as wildcards.

Return Values

Returns a statement resource with a result set containing rows describing the privileges for the tables that match the specified parameters. The rows are composed of the following columns:

Column name	Description
TABLE_CAT	The catalog that contains the table. The value is **NULL** if this table does not have catalogs.
TABLE_SCHEM	Name of the schema that contains the table.
TABLE_NAME	Name of the table.
GRANTOR	Authorization ID of the user who granted the privilege.

GRANTEE	Authorization ID of the user to whom the privilege was granted.
PRIVILEGE	The privilege that has been granted. This can be one of ALTER, CONTROL, DELETE, INDEX, INSERT, REFERENCES, SELECT, or UPDATE.
IS_GRANTABLE	A string value of "YES" or "NO" indicating whether the grantee can grant the privilege to other users.

See Also

- db2_column_privileges()
- db2_columns()
- db2_foreign_keys()
- db2_primary_keys()
- db2_procedure_columns()
- db2_procedures()
- db2_special_columns()
- db2_statistics()
- db2_tables()

db2_tables

db2_tables -- Returns a result set listing the tables and associated metadata in a database

Description

resource **db2_tables** (resource `$connection` [, string `$qualifier` [, string `$schema` [, string `$table-name` [, string `$table-type`]]]])

Returns a result set listing the tables and associated metadata in a database.

Parameters

`connection`
A valid connection to an IBM DB2, Cloudscape, or Apache Derby database.

`qualifier`
A qualifier for DB2 databases running on OS/390 or z/OS servers. For other databases, pass **NULL** or an empty string.

`schema`
The schema which contains the tables. This parameter accepts a search pattern containing _ and *%* as wildcards.

`table-name`
The name of the table. This parameter accepts a search pattern containing _ and *%* as wildcards.

`table-type`
A list of comma-delimited table type identifiers. To match all table types, pass **NULL** or an empty string. Valid table type identifiers include: ALIAS, HIERARCHY TABLE, INOPERATIVE VIEW, NICKNAME, MATERIALIZED QUERY TABLE, SYSTEM TABLE, TABLE, TYPED TABLE, TYPED VIEW, and VIEW.

Return Values

Returns a statement resource with a result set containing rows describing the tables that match the specified parameters. The rows are composed of the following columns:

Column name	Description
TABLE_CAT	The catalog that contains the table. The value is **NULL** if this table does not have catalogs.
TABLE_SCHEM	Name of the schema that contains the table.

TABLE_NAME	Name of the table.
TABLE_TYPE	Table type identifier for the table.
REMARKS	Description of the table.

See Also

- db2_column_privileges()
- db2_columns()
- db2_foreign_keys()
- db2_primary_keys()
- db2_procedure_columns()
- db2_procedures()
- db2_special_columns()
- db2_statistics()
- db2_table_privileges()