MySQL and PHP

MySQL and PHP Reference

Abstract

This manual describes the PHP extensions and interfaces that can be used with MySQL.

Document generated on: 2010-09-22 (revision: 22813)

If you want help with using MySQL, please visit either the MySQL Forums or MySQL Mailing Lists where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML and PDF formats, see the MySQL Documentation Library.

Preface

PHP is a server-side, HTML-embedded scripting language that may be used to create dynamic Web pages. It is available for most operating systems and Web servers, and can access most common databases, including MySQL. PHP may be run as a separate program or compiled as a module for use with the Apache Web server.

PHP actually provides two different MySQL API extensions:

- `mysql`: Available for PHP versions 4 and 5, this extension is intended for use with MySQL versions prior to MySQL 4.1. This extension does not support the improved authentication protocol used in MySQL 4.1, nor does it support prepared statements or multiple statements. If you wish to use this extension with MySQL 4.1, you will likely want to configure the MySQL server to use the `--old-passwords` option (see `Client does not support authentication protocol`). This extension is documented on the PHP Web site at http://php.net/mysql.

- Chapter 2, *MySQL Improved Extension (`Mysqli`)* - Stands for “MySQL, Improved”; this extension is available only in PHP 5. It is intended for use with MySQL 4.1.1 and later. This extension fully supports the authentication protocol used in MySQL 5.0, as well as the Prepared Statements and Multiple Statements APIs. In addition, this extension provides an advanced, object-oriented programming interface. You can read the documentation for the `mysqli` extension at http://php.net/mysqli. Helpful article can be found at http://devzone.zend.com/node/view/id/686 and http://devzone.zend.com/node/view/id/687.

If you're experiencing problems with enabling both the `mysql` and the `mysqli` extension when building PHP on Linux yourself, see Chapter 7, *Enabling Both `mysql` and `mysqli` in PHP*.

The PHP distribution and documentation are available from the PHP Web site.

Chapter 1. MySQL Extension (mysql)

These functions allow you to access MySQL database servers. More information about MySQL can be found at ht-tp://www.mysql.com/.

Documentation for MySQL can be found at http://dev.mysql.com/doc/.

For an overview of MySQL database connectivity terms and products see Section 2.2, “Overview”.

1.1. Installing/Configuring

1.1.1. Requirements

In order to have these functions available, you must compile PHP with MySQL support.

1.1.2. Installation

For compiling, simply use the `--with-mysql[=DIR]` configuration option where the optional `[DIR]` points to the MySQL installation directory.

Although this MySQL extension is compatible with MySQL 4.1.0 and greater, it doesn't support the extra functionality that these versions provide. For that, use the MySQLi extension.

If you would like to install the mysql extension along with the mysqli extension you have to use the same client library to avoid any conflicts.

1.1.2.1. Installation on Linux Systems

1.1.2.1.1. PHP 4

The option `--with-mysql` is enabled by default. This default behavior may be disabled with the `--without-mysql` configure option. If MySQL is enabled without specifying the path to the MySQL install DIR, PHP will use the bundled MySQL client libraries.

Users who run other applications that use MySQL (for example, `auth-mysql`) should not use the bundled library, but rather specify the path to MySQL's install directory, like so: `--with-mysql=/path/to/mysql`. This will force PHP to use the client libraries installed by MySQL, thus avoiding any conflicts.

1.1.2.1.2. PHP 5.0.x, 5.1.x, 5.2.x

MySQL is not enabled by default, nor is the MySQL library bundled with PHP. Read this FAQ for details on why. Use the `--with-mysql[=DIR]` configure option to include MySQL support. You can download *headers and libraries* from ht-tp://www.mysql.com/.

1.1.2.1.3. PHP 5.3.0+

In PHP 5.3.0 and above the MySQL-related database extensions use the MySQL Native Driver by default. This means that the MySQL

Client Library (`libmysql`) is no longer required in order to support connection to a MySQL database. The extensions `mysql`, `mysqli`, and `PHP_PDO_MYSQL` are all enabled by default in PHP 5.3.0+, and all use the MySQL Native Driver by default. In each case no further installation steps are required in order to use these extensions, although you may want to set some preferences in `php.ini` depending on your requirements.

1.1.2.2. Installation on Windows Systems

1.1.2.2.1. PHP 4

The PHP MySQL extension is compiled into PHP.

1.1.2.2.2. PHP 5.0.x, 5.1.x, 5.2.x

MySQL is no longer enabled by default, so the `php_mysql.dll` DLL must be enabled inside of `php.ini`. Also, PHP needs access to the MySQL client library. A file named `libmysql.dll` is included in the Windows PHP distribution and in order for PHP to talk to MySQL this file needs to be available to the Windows systems `PATH`. See the FAQ titled "How do I add my PHP directory to the `PATH` on Windows" for information on how to do this. Although copying `libmysql.dll` to the Windows system directory also works (because the system directory is by default in the system's `PATH`), it's not recommended.

As with enabling any PHP extension (such as `php_mysql.dll`), the PHP directive extension_dir should be set to the directory where the PHP extensions are located. See also the Manual Windows Installation Instructions. An example extension_dir value for PHP 5 is `c:\php\ext`

Note

If when starting the web server an error similar to the following occurs: `"Unable to load dynamic library './php_mysql.dll'"`, this is because `php_mysql.dll` and/or `libmysql.dll` cannot be found by the system.

1.1.2.2.3. PHP 5.3.0+

Please refer to these notes on installing MySQL support on PHP 5.3.0 and above.

1.1.2.3. MySQL Installation Notes

Warning

Crashes and startup problems of PHP may be encountered when loading this extension in conjunction with the recode extension. See the recode extension for more information.

Note

If you need charsets other than *latin* (default), you have to install external (not bundled) libmysql with compiled charset support.

1.1.3. Runtime Configuration

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

Table 1.1. MySQL Configuration Options

Name	Default	Changeable	Changelog
mysql.allow_persistent	"1"	PHP_INI_SYSTEM	
mysql.max_persistent	"-1"	PHP_INI_SYSTEM	
mysql.max_links	"-1"	PHP_INI_SYSTEM	
mysql.trace_mode	"0"	PHP_INI_ALL	Available since PHP 4.3.0.
mysql.default_port	NULL	PHP_INI_ALL	
mysql.default_socket	NULL	PHP_INI_ALL	Available since PHP 4.0.1.
mysql.default_host	NULL	PHP_INI_ALL	
mysql.default_user	NULL	PHP_INI_ALL	
mysql.default_password	NULL	PHP_INI_ALL	
mysql.connect_timeout	"60"	PHP_INI_ALL	PHP_INI_SYSTEM in PHP <= 4.3.2. Available since PHP 4.3.0.

For further details and definitions of the PHP_INI_* modes, see the configuration.changes.modes.

Here's a short explanation of the configuration directives.

`mysql.allow_persistent` boolean	Whether to allow persistent connections to MySQL.
`mysql.max_persistent` integer	The maximum number of persistent MySQL connections per process.
`mysql.max_links` integer	The maximum number of MySQL connections per process, including persistent connections.
`mysql.trace_mode` boolean	Trace mode. When `mysql.trace_mode` is enabled, warnings for table/index scans, non free result sets, and SQL-Errors will be displayed. (Introduced in PHP 4.3.0)
`mysql.default_port` string	The default TCP port number to use when connecting to the database server if no other port is specified. If no default is specified, the port will be obtained from the `MYSQL_TCP_PORT` environment variable, the `mysql-tcp` entry in `/etc/services` or the compile-time `MYSQL_PORT` constant, in that order. Win32 will only use the `MYSQL_PORT` constant.
`mysql.default_socket` string	The default socket name to use when connecting to a local database server if no other socket name is specified.
`mysql.default_host` string	The default server host to use when connecting to the database server if no other host is specified. Doesn't apply in SQL safe mode.
`mysql.default_user` string	The default user name to use when connecting to the database server if no other name is specified. Doesn't apply in SQL safe mode.
`mysql.default_password` string	The default password to use when connecting to the database server if no other password is specified. Doesn't apply in SQL safe mode.
`mysql.connect_timeout` integer	Connect timeout in seconds. On Linux this timeout is also used for waiting for the first answer from the server.

1.1.4. Resource Types

There are two resource types used in the MySQL module. The first one is the link identifier for a database connection, the second a resource which holds the result of a query.

1.2. 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.

Since PHP 4.3.0 it is possible to specify additional client flags for the `mysql_connect` and `mysql_pconnect` functions. The following constants are defined:

Table 1.2. MySQL client constants

Constant	Description
`MYSQL_CLIENT_COMPRESS`	Use compression protocol
`MYSQL_CLIENT_IGNORE_SPACE`	Allow space after function names
`MYSQL_CLIENT_INTERACTIVE`	Allow interactive_timeout seconds (instead of `wait_timeout`) of inactivity before closing the connection.
`MYSQL_CLIENT_SSL`	Use SSL encryption. This flag is only available with version 4.x of the MySQL client library or newer. Version 3.23.x is bundled both with PHP 4 and Windows binaries of PHP 5.

The function `mysql_fetch_array` uses a constant for the different types of result arrays. The following constants are defined:

Table 1.3. MySQL fetch constants

Constant	Description
`MYSQL_ASSOC`	Columns are returned into the array having the fieldname as the array index.
`MYSQL_BOTH`	Columns are returned into the array having both a numerical index and the fieldname as the array index.
`MYSQL_NUM`	Columns are returned into the array having a numerical index to the fields. This index starts with 0, the first field in the result.

1.3. Examples

1.3.1. Basic

This simple example shows how to connect, execute a query, print resulting rows and disconnect from a MySQL database.

Example 1.1. MySQL extension overview example

```
<?php
// Connecting, selecting database
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
    or die('Could not connect: ' . mysql_error());
echo 'Connected successfully';
mysql_select_db('my_database') or die('Could not select database');
// Performing SQL query
$query = 'SELECT * FROM my_table';
$result = mysql_query($query) or die('Query failed: ' . mysql_error());
// Printing results in HTML
echo "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
}
echo "</table>\n";
// Free resultset
```

```
mysql_free_result($result);
// Closing connection
mysql_close($link);
?>
```

1.4. MySQL Functions

Note

Most MySQL functions accept *`link_identifier`* as the last optional parameter. If it is not provided, last opened connection is used. If it doesn't exist, connection is tried to establish with default parameters defined in `php.ini`. If it is not successful, functions return `FALSE` .

1.4.1. `mysql_affected_rows`

- `mysql_affected_rows`

  Get number of affected rows in previous MySQL operation

Description

```
 int mysql_affected_rows(resource link_identifier);
```

Get the number of affected rows by the last INSERT, UPDATE, REPLACE or DELETE query associated with *`link_identifier`*.

Parameters

`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns the number of affected rows on success, and -1 if the last query failed.

If the last query was a DELETE query with no WHERE clause, all of the records will have been deleted from the table but this function will return zero with MySQL versions prior to 4.1.2.

When using UPDATE, MySQL will not update columns where the new value is the same as the old value. This creates the possibility that `mysql_affected_rows` may not actually equal the number of rows matched, only the number of rows that were literally affected by the query.

The REPLACE statement first deletes the record with the same primary key and then inserts the new record. This function returns the number of deleted records plus the number of inserted records.

Examples

Example 1.2. `mysql_affected_rows` example

```
<?php
```

```
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
mysql_select_db('mydb');
/* this should return the correct numbers of deleted records */
mysql_query('DELETE FROM mytable WHERE id < 10');
printf("Records deleted: %d\n", mysql_affected_rows());
/* with a where clause that is never true, it should return 0 */
mysql_query('DELETE FROM mytable WHERE 0');
printf("Records deleted: %d\n", mysql_affected_rows());
?>
```

The above example will output something similar to:

```
Records deleted: 10
Records deleted: 0
```

Example 1.3. `mysql_affected_rows` example using transactions

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
mysql_select_db('mydb');
/* Update records */
mysql_query("UPDATE mytable SET used=1 WHERE id < 10");
printf ("Updated records: %d\n", mysql_affected_rows());
mysql_query("COMMIT");
?>
```

The above example will output something similar to:

```
Updated Records: 10
```

Notes

Transactions

If you are using transactions, you need to call `mysql_affected_rows` after your INSERT, UPDATE, or DELETE query, not after the COMMIT.

SELECT Statements

To retrieve the number of rows returned by a SELECT, it is possible to use `mysql_num_rows`.

Cascaded Foreign Keys

`mysql_affected_rows` does not count rows affected implicitly through the use of ON DELETE CASCADE and/or ON UPDATE CASCADE in foreign key constraints.

See Also

`mysql_num_rows`

mysql_info

1.4.2. mysql_client_encoding

- mysql_client_encoding

  Returns the name of the character set

Description

```
string mysql_client_encoding(resource link_identifier);
```

Retrieves the `character_set` variable from MySQL.

Parameters

link_identifier	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns the default character set name for the current connection.

Examples

Example 1.4. mysql_client_encoding example

```
<?php
$link    = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$charset = mysql_client_encoding($link);
echo "The current character set is: $charset\n";
?>
```

The above example will output something similar to:

```
The current character set is: latin1
```

See Also

mysql_set_charset
mysql_real_escape_string

1.4.3. mysql_close

- mysql_close

Close MySQL connection

Description

```
bool mysql_close(resource link_identifier);
```

`mysql_close` closes the non-persistent connection to the MySQL server that's associated with the specified link identifier. If *`link_identifier`* isn't specified, the last opened link is used.

Using `mysql_close` isn't usually necessary, as non-persistent open links are automatically closed at the end of the script's execution. See also freeing resources.

Parameters

`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

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

Examples

Example 1.5. `mysql_close` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
?>
```

The above example will output:

```
Connected successfully
```

Notes

Note

`mysql_close` will not close persistent links created by `mysql_pconnect`.

See Also

`mysql_connect`
`mysql_free_result`

1.4.4. `mysql_connect`

- `mysql_connect`

  Open a connection to a MySQL Server

Description

```
resource mysql_connect(string server= =ini_get("mysql.default_host"),
                       string username= =ini_get("mysql.default_user"),
                       string password= =ini_get("mysql.default_password"),
                       bool new_link= =false,
                       int client_flags= =0);
```

Opens or reuses a connection to a MySQL server.

Parameters

`server`	The MySQL server. It can also include a port number. e.g. "hostname:port" or a path to a local socket e.g. ":/path/to/socket" for the localhost. If the PHP directive mysql.default_host is undefined (default), then the default value is 'localhost:3306'. In SQL safe mode, this parameter is ignored and value 'localhost:3306' is always used.
`username`	The username. Default value is defined by mysql.default_user. In SQL safe mode, this parameter is ignored and the name of the user that owns the server process is used.
`password`	The password. Default value is defined by mysql.default_password. In SQL safe mode, this parameter is ignored and empty password is used.
`new_link`	If a second call is made to `mysql_connect` with the same arguments, no new link will be established, but instead, the link identifier of the already opened link will be returned. The *`new_link`* parameter modifies this behavior and makes `mysql_connect` always open a new link, even if `mysql_connect` was called before with the same parameters. In SQL safe mode, this parameter is ignored.
`client_flags`	The *`client_flags`* parameter can be a combination of the following constants: 128 (enable `LOAD DATA LOCAL` handling), `MYSQL_CLIENT_SSL` , `MYSQL_CLIENT_COMPRESS` , `MYSQL_CLIENT_IGNORE_SPACE` or `MYSQL_CLIENT_INTERACTIVE` . Read the section about Table 1.2, “MySQL client constants” for further information. In SQL safe mode, this parameter is ignored.

Return Values

Returns a MySQL link identifier on success or `FALSE` on failure.

Changelog

Version	Description
4.3.0	Added the *`client_flags`* parameter.
4.2.0	Added the *`new_link`* parameter.

Examples

Example 1.6. `mysql_connect` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
?>
```

Example 1.7. `mysql_connect` example using `hostname:port` syntax

```
<?php
// we connect to example.com and port 3307
$link = mysql_connect('example.com:3307', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
// we connect to localhost at port 3307
$link = mysql_connect('127.0.0.1:3307', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
?>
```

Example 1.8. `mysql_connect` example using ":/path/to/socket" syntax

```
<?php
// we connect to localhost and socket e.g. /tmp/mysql.sock
//variant 1: ommit localhost
$link = mysql_connect(':/tmp/mysql', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
// variant 2: with localhost
$link = mysql_connect('localhost:/tmp/mysql.sock', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
?>
```

Notes

Note

Whenever you specify "localhost" or "localhost:port" as server, the MySQL client library will override this and try to connect to a local socket (named pipe on Windows). If you want to use TCP/IP, use "127.0.0.1" instead of "localhost". If the MySQL client library tries to connect to the wrong local socket, you should set the correct path as *`mysql.default_host`* string in your PHP configuration and leave the server field blank.

Note

The link to the server will be closed as soon as the execution of the script ends, unless it's closed earlier by explicitly calling `mysql_close`.

Note

You can suppress the error message on failure by prepending a @ to the function name.

Note

Error "Can't create TCP/IP socket (10106)" usually means that the variables_order configure directive doesn't contain character `E`. On Windows, if the environment is not copied the `SYSTEMROOT` environment variable won't be available and PHP will have problems loading Winsock.

See Also

`mysql_pconnect`
`mysql_close`

1.4.5. `mysql_create_db`

- `mysql_create_db`

  Create a MySQL database

Description

```
bool mysql_create_db(string database_name,
                     resource link_identifier);
```

`mysql_create_db` attempts to create a new database on the server associated with the specified link identifier.

Parameters

`database_name`	The name of the database being created.
`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

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

Examples

Example 1.9. `mysql_create_db` alternative example

The function `mysql_create_db` is deprecated. It is preferable to use `mysql_query` to issue an sql `CREATE DATABASE` statement instead.

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
$sql = 'CREATE DATABASE my_db';
if (mysql_query($sql, $link)) {
    echo "Database my_db created successfully\n";
} else {
    echo 'Error creating database: ' . mysql_error() . "\n";
}
?>
```

The above example will output something similar to:

```
Database my_db created successfully
```

Notes

> **Note**
>
> For backward compatibility, the following deprecated alias may be used: `mysql_createdb`

> **Note**
>
> This function will not be available if the MySQL extension was built against a MySQL 4.x client library.

See Also

`mysql_query`
`mysql_select_db`

1.4.6. `mysql_data_seek`

- `mysql_data_seek`

  Move internal result pointer

Description

```
  bool mysql_data_seek(resource result,
                       int row_number);
```

`mysql_data_seek` moves the internal row pointer of the MySQL result associated with the specified result identifier to point to the specified row number. The next call to a MySQL fetch function, such as `mysql_fetch_assoc`, would return that row.

`row_number` starts at 0. The *`row_number`* should be a value in the range from 0 to `mysql_num_rows` - 1. However if the result set is empty (`mysql_num_rows` == 0), a seek to 0 will fail with a E_WARNING and `mysql_data_seek` will return `FALSE` .

Parameters

`result`	The result resource that is being evaluated. This result comes from a call to `mysql_query`.
`row_number`	The desired row number of the new result pointer.

Return Values

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

Examples

Example 1.10. `mysql_data_seek` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
$db_selected = mysql_select_db('sample_db');
if (!$db_selected) {
    die('Could not select database: ' . mysql_error());
}
$query = 'SELECT last_name, first_name FROM friends';
$result = mysql_query($query);
if (!$result) {
    die('Query failed: ' . mysql_error());
}
/* fetch rows in reverse order */
for ($i = mysql_num_rows($result) - 1; $i >= 0; $i--) {
    if (!mysql_data_seek($result, $i)) {
        echo "Cannot seek to row $i: " . mysql_error() . "\n";
        continue;
    }
    if (!($row = mysql_fetch_assoc($result))) {
        continue;
    }
    echo $row['last_name'] . ' ' . $row['first_name'] . "<br />\n";
}
mysql_free_result($result);
?>
```

Notes

Note

The function `mysql_data_seek` can be used in conjunction only with `mysql_query`, not with `mysql_unbuffered_query`.

See Also

`mysql_query`
`mysql_num_rows`
`mysql_fetch_row`
`mysql_fetch_assoc`
`mysql_fetch_array`
`mysql_fetch_object`

1.4.7. `mysql_db_name`

- `mysql_db_name`

  Get result data

Description

```
  string mysql_db_name(resource result,
                       int row,
                       mixed field);
```

Retrieve the database name from a call to `mysql_list_dbs`.

Parameters

result	The result pointer from a call to mysql_list_dbs.
row	The index into the result set.
field	The field name.

Return Values

Returns the database name on success, and FALSE on failure. If FALSE is returned, use mysql_error to determine the nature of the error.

Examples

Example 1.11. mysql_db_name example

```
<?php
error_reporting(E_ALL);
$link = mysql_connect('dbhost', 'username', 'password');
$db_list = mysql_list_dbs($link);
$i = 0;
$cnt = mysql_num_rows($db_list);
while ($i < $cnt) {
    echo mysql_db_name($db_list, $i) . "\n";
    $i++;
}
?>
```

Notes

Note

For backward compatibility, the following deprecated alias may be used: mysql_dbname

See Also

mysql_list_dbs
mysql_tablename

1.4.8. mysql_db_query

- mysql_db_query

  Send a MySQL query

Description

```
  resource mysql_db_query(string database,
                          string query,
                          resource link_identifier);
```

mysql_db_query selects a database, and executes a query on it.

Warning

This function has been *DEPRECATED* as of PHP 5.3.0. Relying on this feature is highly discouraged.

Parameters

`database`	The name of the database that will be selected.
`query`	The MySQL query. Data inside the query should be properly escaped.
`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns a positive MySQL result resource to the query result, or `FALSE` on error. The function also returns `TRUE` / `FALSE` for `INSERT`/`UPDATE`/`DELETE` queries to indicate success/failure.

Changelog

Version	Description
5.3.0	This function now throws an E_DEPRECATED notice.
4.0.6	This function is deprecated, do not use this function. Use `mysql_select_db` and `mysql_query` instead.

Examples

Example 1.12. `mysql_db_query` alternative example

```
<?php
if (!$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')) {
    echo 'Could not connect to mysql';
    exit;
}
if (!mysql_select_db('mysql_dbname', $link)) {
    echo 'Could not select database';
    exit;
}
$sql    = 'SELECT foo FROM bar WHERE id = 42';
$result = mysql_query($sql, $link);
if (!$result) {
    echo "DB Error, could not query the database\n";
    echo 'MySQL Error: ' . mysql_error();
    exit;
}
while ($row = mysql_fetch_assoc($result)) {
    echo $row['foo'];
}
mysql_free_result($result);
?>
```

Notes

Note

Be aware that this function does *NOT* switch back to the database you were connected before. In other words, you can't use this function to *temporarily* run a sql query on another database, you would have to manually switch back. Users are strongly encouraged to use the `database.table` syntax in their sql queries or `mysql_select_db` instead of this function.

See Also

`mysql_query`
`mysql_select_db`

1.4.9. `mysql_drop_db`

- `mysql_drop_db`

  Drop (delete) a MySQL database

Description

```
  bool mysql_drop_db(string database_name,
                     resource link_identifier);
```

`mysql_drop_db` attempts to drop (remove) an entire database from the server associated with the specified link identifier. This function is deprecated, it is preferable to use `mysql_query` to issue an sql `DROP DATABASE` statement instead.

Parameters

`database_name`	The name of the database that will be deleted.
`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

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

Examples

Example 1.13. `mysql_drop_db` alternative example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
$sql = 'DROP DATABASE my_db';
if (mysql_query($sql, $link)) {
    echo "Database my_db was successfully dropped\n";
} else {
    echo 'Error dropping database: ' . mysql_error() . "\n";
}
?>
```

Notes

Warning

This function will not be available if the MySQL extension was built against a MySQL 4.x client library.

Note

For backward compatibility, the following deprecated alias may be used: `mysql_dropdb`

See Also

`mysql_query`

1.4.10. `mysql_errno`

- `mysql_errno`

  Returns the numerical value of the error message from previous MySQL operation

Description

```
int mysql_errno(resource link_identifier);
```

Returns the error number from the last MySQL function.

Errors coming back from the MySQL database backend no longer issue warnings. Instead, use `mysql_errno` to retrieve the error code. Note that this function only returns the error code from the most recently executed MySQL function (not including `mysql_error` and `mysql_errno`), so if you want to use it, make sure you check the value before calling another MySQL function.

Parameters

`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns the error number from the last MySQL function, or `0` (zero) if no error occurred.

Examples

Example 1.14. `mysql_errno` example

```
<?php
$link = mysql_connect("localhost", "mysql_user", "mysql_password");
if (!mysql_select_db("nonexistentdb", $link)) {
    echo mysql_errno($link) . ": " . mysql_error($link). "\n";
}
mysql_select_db("kossu", $link);
if (!mysql_query("SELECT * FROM nonexistenttable", $link)) {
    echo mysql_errno($link) . ": " . mysql_error($link) . "\n";
}
?>
```

The above example will output something similar to:

```
1049: Unknown database 'nonexistentdb'
1146: Table 'kossu.nonexistenttable' doesn't exist
```

See Also

mysql_error
MySQL error codes

1.4.11. mysql_error

- mysql_error

  Returns the text of the error message from previous MySQL operation

Description

```
string mysql_error(resource link_identifier);
```

Returns the error text from the last MySQL function. Errors coming back from the MySQL database backend no longer issue warnings. Instead, use `mysql_error` to retrieve the error text. Note that this function only returns the error text from the most recently executed MySQL function (not including `mysql_error` and `mysql_errno`), so if you want to use it, make sure you check the value before calling another MySQL function.

Parameters

link_identifier	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns the error text from the last MySQL function, or `''` (empty string) if no error occurred.

Examples

Example 1.15. mysql_error example

```
<?php
$link = mysql_connect("localhost", "mysql_user", "mysql_password");
mysql_select_db("nonexistentdb", $link);
echo mysql_errno($link) . ": " . mysql_error($link). "\n";
mysql_select_db("kossu", $link);
mysql_query("SELECT * FROM nonexistenttable", $link);
echo mysql_errno($link) . ": " . mysql_error($link) . "\n";
?>
```

The above example will output something similar to:

```
1049: Unknown database 'nonexistentdb'
1146: Table 'kossu.nonexistenttable' doesn't exist
```

See Also

`mysql_errno`
MySQL error codes

1.4.12. `mysql_escape_string`

- `mysql_escape_string`

  Escapes a string for use in a mysql_query

Description

```
string mysql_escape_string(string unescaped_string);
```

This function will escape the *`unescaped_string`*, so that it is safe to place it in a `mysql_query`. This function is deprecated.

This function is identical to `mysql_real_escape_string` except that `mysql_real_escape_string` takes a connection handler and escapes the string according to the current character set. `mysql_escape_string` does not take a connection argument and does not respect the current charset setting.

Warning

This function has been *DEPRECATED* as of PHP 5.3.0. Relying on this feature is highly discouraged.

Parameters

`unescaped_string`	The string that is to be escaped.

Return Values

Returns the escaped string.

Changelog

Version	Description
5.3.0	This function now throws an E_DEPRECATED notice.
4.3.0	This function became deprecated, do not use this function. Instead, use `mysql_real_escape_string`.

Examples

Example 1.16. `mysql_escape_string` example

```
<?php
$item = "Zak's Laptop";
$escaped_item = mysql_escape_string($item);
printf("Escaped string: %s\n", $escaped_item);
?>
```

The above example will output:

```
Escaped string: Zak\'s Laptop
```

Notes

Note

`mysql_escape_string` does not escape `%` and `_`.

See Also

`mysql_real_escape_string`
`addslashes`
The magic_quotes_gpc directive.

1.4.13. `mysql_fetch_array`

- `mysql_fetch_array`

  Fetch a result row as an associative array, a numeric array, or both

Description

```
array mysql_fetch_array(resource result,
                        int result_type= =MYSQL_BOTH);
```

Returns an array that corresponds to the fetched row and moves the internal data pointer ahead.

Parameters

`result`	The result resource that is being evaluated. This result comes from a call to `mysql_query`.
`result_type`	The type of array that is to be fetched. It's a constant and can take the following values: `MYSQL_ASSOC` , `MYSQL_NUM` , and `MYSQL_BOTH` .

Return Values

Returns an array of strings that corresponds to the fetched row, or `FALSE` if there are no more rows. The type of returned array depends on how *`result_type`* is defined. By using `MYSQL_BOTH` (default), you'll get an array with both associative and number indices. Using `MYSQL_ASSOC` , you only get associative indices (as `mysql_fetch_assoc` works), using `MYSQL_NUM` , you only get number indices (as `mysql_fetch_row` works).

If two or more columns of the result have the same field names, the last column will take precedence. To access the other column(s) of the same name, you must use the numeric index of the column or make an alias for the column. For aliased columns, you cannot access the contents with the original column name.

Examples

Example 1.17. Query with aliased duplicate field names

```
SELECT table1.field AS foo, table2.field AS bar FROM table1, table2
```

Example 1.18. `mysql_fetch_array` with `MYSQL_NUM`

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("Could not connect: " . mysql_error());
mysql_select_db("mydb");
$result = mysql_query("SELECT id, name FROM mytable");
while ($row = mysql_fetch_array($result, MYSQL_NUM)) {
    printf("ID: %s  Name: %s", $row[0], $row[1]);
}
mysql_free_result($result);
?>
```

Example 1.19. `mysql_fetch_array` with `MYSQL_ASSOC`

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("Could not connect: " . mysql_error());
mysql_select_db("mydb");
$result = mysql_query("SELECT id, name FROM mytable");
while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
    printf("ID: %s  Name: %s", $row["id"], $row["name"]);
}
mysql_free_result($result);
?>
```

Example 1.20. `mysql_fetch_array` with `MYSQL_BOTH`

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("Could not connect: " . mysql_error());
mysql_select_db("mydb");
$result = mysql_query("SELECT id, name FROM mytable");
while ($row = mysql_fetch_array($result, MYSQL_BOTH)) {
    printf ("ID: %s  Name: %s", $row[0], $row["name"]);
}
mysql_free_result($result);
?>
```

Notes

Performance

An important thing to note is that using `mysql_fetch_array` is *not significantly* slower than using `mysql_fetch_row`, while it provides a significant added value.

Note

Field names returned by this function are *case-sensitive*.

Note

This function sets NULL fields to the PHP `NULL` value.

See Also

`mysql_fetch_row`
`mysql_fetch_assoc`
`mysql_data_seek`
`mysql_query`

1.4.14. `mysql_fetch_assoc`

- `mysql_fetch_assoc`

  Fetch a result row as an associative array

Description

```
array mysql_fetch_assoc(resource result);
```

Returns an associative array that corresponds to the fetched row and moves the internal data pointer ahead. `mysql_fetch_assoc` is equivalent to calling `mysql_fetch_array` with MYSQL_ASSOC for the optional second parameter. It only returns an associative array.

Parameters

`result`	The result resource that is being evaluated. This result comes from a call to `mysql_query`.

Return Values

Returns an associative array of strings that corresponds to the fetched row, or `FALSE` if there are no more rows.

If two or more columns of the result have the same field names, the last column will take precedence. To access the other column(s) of the same name, you either need to access the result with numeric indices by using `mysql_fetch_row` or add alias names. See the example at the `mysql_fetch_array` description about aliases.

Examples

Example 1.21. An expanded `mysql_fetch_assoc` example

```
<?php
$conn = mysql_connect("localhost", "mysql_user", "mysql_password");
if (!$conn) {
    echo "Unable to connect to DB: " . mysql_error();
    exit;
}

if (!mysql_select_db("mydbname")) {
    echo "Unable to select mydbname: " . mysql_error();
    exit;
}
$sql = "SELECT id as userid, fullname, userstatus
        FROM   sometable
        WHERE  userstatus = 1";
$result = mysql_query($sql);
if (!$result) {
    echo "Could not successfully run query ($sql) from DB: " . mysql_error();
    exit;
}
if (mysql_num_rows($result) == 0) {
    echo "No rows found, nothing to print so am exiting";
    exit;
}
// While a row of data exists, put that row in $row as an associative array
// Note: If you're expecting just one row, no need to use a loop
// Note: If you put extract($row); inside the following loop, you'll
//       then create $userid, $fullname, and $userstatus
while ($row = mysql_fetch_assoc($result)) {
    echo $row["userid"];
    echo $row["fullname"];
    echo $row["userstatus"];
}
mysql_free_result($result);
?>
```

Notes

> **Performance**
>
> An important thing to note is that using `mysql_fetch_assoc` is *not significantly* slower than using `mysql_fetch_row`, while it provides a significant added value.

> **Note**
>
> Field names returned by this function are *case-sensitive*.

> **Note**
>
> This function sets NULL fields to the PHP `NULL` value.

See Also

`mysql_fetch_row`
`mysql_fetch_array`
`mysql_data_seek`
`mysql_query`
`mysql_error`

1.4.15. `mysql_fetch_field`

- `mysql_fetch_field`

  Get column information from a result and return as an object

Description

```
 object mysql_fetch_field(resource result,
                          int field_offset= =0);
```

Returns an object containing field information. This function can be used to obtain information about fields in the provided query result.

Parameters

`result`	The result resource that is being evaluated. This result comes from a call to `mysql_query`.
`field_offset`	The numerical field offset. If the field offset is not specified, the next field that was not yet retrieved by this function is retrieved. The *`field_offset`* starts at `0`.

Return Values

Returns an object containing field information. The properties of the object are:

- name - column name
- table - name of the table the column belongs to
- max_length - maximum length of the column
- not_null - 1 if the column cannot be NULL
- primary_key - 1 if the column is a primary key
- unique_key - 1 if the column is a unique key
- multiple_key - 1 if the column is a non-unique key
- numeric - 1 if the column is numeric
- blob - 1 if the column is a BLOB
- type - the type of the column
- unsigned - 1 if the column is unsigned
- zerofill - 1 if the column is zero-filled

Examples

Example 1.22. mysql_fetch_field example

```
<?php
$conn = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$conn) {
    die('Could not connect: ' . mysql_error());
}
mysql_select_db('database');
$result = mysql_query('select * from table');
if (!$result) {
    die('Query failed: ' . mysql_error());
}
/* get column metadata */
$i = 0;
while ($i < mysql_num_fields($result)) {
    echo "Information for column $i:<br />\n";
    $meta = mysql_fetch_field($result, $i);
    if (!$meta) {
        echo "No information available<br />\n";
    }
    echo "<pre>
blob:         $meta->blob
max_length:   $meta->max_length
multiple_key: $meta->multiple_key
name:         $meta->name
not_null:     $meta->not_null
numeric:      $meta->numeric
primary_key:  $meta->primary_key
table:        $meta->table
type:         $meta->type
unique_key:   $meta->unique_key
unsigned:     $meta->unsigned
zerofill:     $meta->zerofill
</pre>";
    $i++;
}
mysql_free_result($result);
?>
```

Notes

> **Note**
>
> Field names returned by this function are *case-sensitive*.

See Also

`mysql_field_seek`

1.4.16. `mysql_fetch_lengths`

- `mysql_fetch_lengths`

  Get the length of each output in a result

Description

```
array mysql_fetch_lengths(resource result);
```

Returns an array that corresponds to the lengths of each field in the last row fetched by MySQL.

`mysql_fetch_lengths` stores the lengths of each result column in the last row returned by `mysql_fetch_row`, `mysql_fetch_assoc`, `mysql_fetch_array`, and `mysql_fetch_object` in an array, starting at offset 0.

Parameters

`result`	The result resource that is being evaluated. This result comes from a call to `mysql_query`.

Return Values

An array of lengths on success or `FALSE` on failure.

Examples

Example 1.23. A `mysql_fetch_lengths` example

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
$row     = mysql_fetch_assoc($result);
$lengths = mysql_fetch_lengths($result);
print_r($row);
print_r($lengths);
?>
```

The above example will output something similar to:

```
Array
(
    [id] => 42
    [email] => user@example.com
)
Array
(
    [0] => 2
    [1] => 16
)
```

See Also

mysql_field_len
mysql_fetch_row
strlen

1.4.17. mysql_fetch_object

- mysql_fetch_object

  Fetch a result row as an object

Description

```
  object mysql_fetch_object(resource result,
                            string class_name,
                            array params);
```

Returns an object with properties that correspond to the fetched row and moves the internal data pointer ahead.

Parameters

result	The result resource that is being evaluated. This result comes from a call to `mysql_query`.
class_name	The name of the class to instantiate, set the properties of and return. If not specified, a `stdClass` object is returned.
params	An optional array of parameters to pass to the constructor for *class_name* objects.

Return Values

Returns an object with string properties that correspond to the fetched row, or `FALSE` if there are no more rows.

Changelog

Version	Description
5.0.0	Added the ability to return as a different object.

Examples

Example 1.24. mysql_fetch_object example

```
<?php
mysql_connect("hostname", "user", "password");
mysql_select_db("mydb");
```

```
$result = mysql_query("select * from mytable");
while ($row = mysql_fetch_object($result)) {
    echo $row->user_id;
    echo $row->fullname;
}
mysql_free_result($result);
?>
```

Example 1.25. mysql_fetch_object example

```
<?php
class foo {
    public $name;
}
mysql_connect("hostname", "user", "password");
mysql_select_db("mydb");
$result = mysql_query("select name from mytable limit 1");
$obj = mysql_fetch_object($result, 'foo');
var_dump($obj);
?>
```

Notes

> **Performance**
>
> Speed-wise, the function is identical to `mysql_fetch_array`, and almost as quick as `mysql_fetch_row` (the difference is insignificant).

> **Note**
>
> `mysql_fetch_object` is similar to `mysql_fetch_array`, with one difference - an object is returned, instead of an array. Indirectly, that means that you can only access the data by the field names, and not by their offsets (numbers are illegal property names).

> **Note**
>
> Field names returned by this function are *case-sensitive*.

> **Note**
>
> This function sets NULL fields to the PHP `NULL` value.

See Also

`mysql_fetch_array`
`mysql_fetch_assoc`
`mysql_fetch_row`
`mysql_data_seek`
`mysql_query`

1.4.18. mysql_fetch_row

- `mysql_fetch_row`

  Get a result row as an enumerated array

Description

```
array mysql_fetch_row(resource result);
```

Returns a numerical array that corresponds to the fetched row and moves the internal data pointer ahead.

Parameters

result	The result resource that is being evaluated. This result comes from a call to mysql_query.

Return Values

Returns an numerical array of strings that corresponds to the fetched row, or FALSE if there are no more rows.

mysql_fetch_row fetches one row of data from the result associated with the specified result identifier. The row is returned as an array. Each result column is stored in an array offset, starting at offset 0.

Examples

Example 1.26. Fetching one row with mysql_fetch_row

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
$row = mysql_fetch_row($result);
echo $row[0]; // 42
echo $row[1]; // the email value
?>
```

Notes

Note

This function sets NULL fields to the PHP NULL value.

See Also

mysql_fetch_array
mysql_fetch_assoc
mysql_fetch_object
mysql_data_seek
mysql_fetch_lengths
mysql_result

1.4.19. mysql_field_flags

- mysql_field_flags

  Get the flags associated with the specified field in a result

Description

```
string mysql_field_flags(resource result,
                         int field_offset);
```

`mysql_field_flags` returns the field flags of the specified field. The flags are reported as a single word per flag separated by a single space, so that you can split the returned value using `explode`.

Parameters

`result`	The result resource that is being evaluated. This result comes from a call to `mysql_query`.
`field_offset`	The numerical field offset. The *`field_offset`* starts at `0`. If *`field_offset`* does not exist, an error of level `E_WARNING` is also issued.

Return Values

Returns a string of flags associated with the result or `FALSE` on failure.

The following flags are reported, if your version of MySQL is current enough to support them: `"not_null"`, `"primary_key"`, `"unique_key"`, `"multiple_key"`, `"blob"`, `"unsigned"`, `"zerofill"`, `"binary"`, `"enum"`, `"auto_increment"` and `"timestamp"`.

Examples

Example 1.27. A `mysql_field_flags` example

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
$flags = mysql_field_flags($result, 0);
echo $flags;
print_r(explode(' ', $flags));
?>
```

The above example will output something similar to:

```
not_null primary_key auto_increment
Array
(
    [0] => not_null
    [1] => primary_key
    [2] => auto_increment
)
```

Notes

Note

For backward compatibility, the following deprecated alias may be used: `mysql_fieldflags`

See Also

`mysql_field_type`
`mysql_field_len`

1.4.20. mysql_field_len

- mysql_field_len

  Returns the length of the specified field

Description

```
int mysql_field_len(resource result,
                    int field_offset);
```

mysql_field_len returns the length of the specified field.

Parameters

result	The result resource that is being evaluated. This result comes from a call to mysql_query.
field_offset	The numerical field offset. The *field_offset* starts at 0. If *field_offset* does not exist, an error of level E_WARNING is also issued.

Return Values

The length of the specified field index on success or FALSE on failure.

Examples

Example 1.28. mysql_field_len example

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
// Will get the length of the id field as specified in the database
// schema.
$length = mysql_field_len($result, 0);
echo $length;
?>
```

Notes

> **Note**
>
> For backward compatibility, the following deprecated alias may be used: mysql_fieldlen

See Also

mysql_fetch_lengths
strlen

1.4.21. mysql_field_name

- `mysql_field_name`

  Get the name of the specified field in a result

Description

```
 string mysql_field_name(resource result,
                         int field_offset);
```

`mysql_field_name` returns the name of the specified field index.

Parameters

`result`	The result resource that is being evaluated. This result comes from a call to `mysql_query`.
`field_offset`	The numerical field offset. The *`field_offset`* starts at `0`. If *`field_offset`* does not exist, an error of level `E_WARNING` is also issued.

Return Values

The name of the specified field index on success or `FALSE` on failure.

Examples

Example 1.29. `mysql_field_name` example

```
<?php
/* The users table consists of three fields:
*   user_id
*   username
*   password.
*/
$link = @mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect to MySQL server: ' . mysql_error());
}
$dbname = 'mydb';
$db_selected = mysql_select_db($dbname, $link);
if (!$db_selected) {
    die("Could not set $dbname: " . mysql_error());
}
$res = mysql_query('select * from users', $link);
echo mysql_field_name($res, 0) . "\n";
echo mysql_field_name($res, 2);
?>
```

The above example will output:

```
user_id
password
```

Notes

> **Note**
>
> Field names returned by this function are *case-sensitive*.

> **Note**

For backward compatibility, the following deprecated alias may be used: `mysql_fieldname`

See Also

`mysql_field_type`
`mysql_field_len`

1.4.22. `mysql_field_seek`

- `mysql_field_seek`

  Set result pointer to a specified field offset

Description

```
bool mysql_field_seek(resource result,
                      int field_offset);
```

Seeks to the specified field offset. If the next call to `mysql_fetch_field` doesn't include a field offset, the field offset specified in `mysql_field_seek` will be returned.

Parameters

`result`	The result resource that is being evaluated. This result comes from a call to `mysql_query`.
`field_offset`	The numerical field offset. The *`field_offset`* starts at `0`. If *`field_offset`* does not exist, an error of level `E_WARNING` is also issued.

Return Values

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

See Also

`mysql_fetch_field`

1.4.23. `mysql_field_table`

- `mysql_field_table`

  Get name of the table the specified field is in

Description

```
string mysql_field_table(resource result,
                         int field_offset);
```

Returns the name of the table that the specified field is in.

Parameters

result	The result resource that is being evaluated. This result comes from a call to `mysql_query`.
field_offset	The numerical field offset. The *field_offset* starts at `0`. If *field_offset* does not exist, an error of level `E_WARNING` is also issued.

Return Values

The name of the table on success.

Examples

Example 1.30. A `mysql_field_table` example

```
<?php
$query = "SELECT account.*, country.* FROM account, country WHERE country.name = 'Portugal' AND account.country_id = count
// get the result from the DB
$result = mysql_query($query);
// Lists the table name and then the field name
for ($i = 0; $i < mysql_num_fields($result); ++$i) {
    $table = mysql_field_table($result, $i);
    $field = mysql_field_name($result, $i);
    echo  "$table: $field\n";
}
?>
```

Notes

> **Note**
>
> For backward compatibility, the following deprecated alias may be used: `mysql_fieldtable`

See Also

`mysql_list_tables`

1.4.24. `mysql_field_type`

Copyright 1997-2010 the PHP Documentation Group.

- `mysql_field_type`

  Get the type of the specified field in a result

Description

```
  string mysql_field_type(resource result,
                          int field_offset);
```

`mysql_field_type` is similar to the `mysql_field_name` function. The arguments are identical, but the field type is returned instead.

Parameters

result	The result resource that is being evaluated. This result comes from a call to `mysql_query`.
field_offset	The numerical field offset. The *field_offset* starts at `0`. If *field_offset* does not exist, an error of level `E_WARNING` is also issued.

Return Values

The returned field type will be one of `"int"`, `"real"`, `"string"`, `"blob"`, and others as detailed in the MySQL documentation.

Examples

Example 1.31. `mysql_field_type` example

```
<?php
mysql_connect("localhost", "mysql_username", "mysql_password");
mysql_select_db("mysql");
$result = mysql_query("SELECT * FROM func");
$fields = mysql_num_fields($result);
$rows   = mysql_num_rows($result);
$table  = mysql_field_table($result, 0);
echo "Your '" . $table . "' table has " . $fields . " fields and " . $rows . " record(s)\n";
echo "The table has the following fields:\n";
for ($i=0; $i < $fields; $i++) {
    $type  = mysql_field_type($result, $i);
    $name  = mysql_field_name($result, $i);
    $len   = mysql_field_len($result, $i);
    $flags = mysql_field_flags($result, $i);
    echo $type . " " . $name . " " . $len . " " . $flags . "\n";
}
mysql_free_result($result);
mysql_close();
?>
```

The above example will output something similar to:

```
Your 'func' table has 4 fields and 1 record(s)
The table has the following fields:
string name 64 not_null primary_key binary
int ret 1 not_null
string dl 128 not_null
string type 9 not_null enum
```

Notes

> **Note**
>
> For backward compatibility, the following deprecated alias may be used: `mysql_fieldtype`

See Also

`mysql_field_name`
`mysql_field_len`

1.4.25. `mysql_free_result`

- `mysql_free_result`

  Free result memory

Description

```
bool mysql_free_result(resource result);
```

`mysql_free_result` will free all memory associated with the result identifier *`result`*.

`mysql_free_result` only needs to be called if you are concerned about how much memory is being used for queries that return large result sets. All associated result memory is automatically freed at the end of the script's execution.

Parameters

`result`	The result resource that is being evaluated. This result comes from a call to `mysql_query`.

Return Values

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

If a non-resource is used for the *`result`*, an error of level E_WARNING will be emitted. It's worth noting that `mysql_query` only returns a resource for SELECT, SHOW, EXPLAIN, and DESCRIBE queries.

Examples

Example 1.32. A `mysql_free_result` example

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
/* Use the result, assuming we're done with it afterwards */
$row = mysql_fetch_assoc($result);
/* Now we free up the result and continue on with our script */
mysql_free_result($result);
echo $row['id'];
echo $row['email'];
?>
```

Notes

Note

For backward compatibility, the following deprecated alias may be used: `mysql_freeresult`

See Also

`mysql_query`
`is_resource`

1.4.26. `mysql_get_client_info`

- `mysql_get_client_info`

  Get MySQL client info

Description

```
string mysql_get_client_info();
```

`mysql_get_client_info` returns a string that represents the client library version.

Return Values

The MySQL client version.

Examples

Example 1.33. `mysql_get_client_info` example

```
<?php
printf("MySQL client info: %s\n", mysql_get_client_info());
?>
```

The above example will output something similar to:

```
MySQL client info: 3.23.39
```

See Also

`mysql_get_host_info`
`mysql_get_proto_info`
`mysql_get_server_info`

1.4.27. `mysql_get_host_info`

- `mysql_get_host_info`

  Get MySQL host info

Description

```
string mysql_get_host_info(resource link_identifier);
```

Describes the type of connection in use for the connection, including the server host name.

Parameters

`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns a string describing the type of MySQL connection in use for the connection or `FALSE` on failure.

Examples

Example 1.34. `mysql_get_host_info` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
printf("MySQL host info: %s\n", mysql_get_host_info());
?>
```

The above example will output something similar to:

```
MySQL host info: Localhost via UNIX socket
```

See Also

`mysql_get_client_info`
`mysql_get_proto_info`
`mysql_get_server_info`

1.4.28. `mysql_get_proto_info`

- `mysql_get_proto_info`

  Get MySQL protocol info

Description

```
  int mysql_get_proto_info(resource link_identifier);
```

Retrieves the MySQL protocol.

Parameters

`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns the MySQL protocol on success or `FALSE` on failure.

Examples

Example 1.35. `mysql_get_proto_info` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
printf("MySQL protocol version: %s\n", mysql_get_proto_info());
?>
```

The above example will output something similar to:

```
MySQL protocol version: 10
```

See Also

`mysql_get_client_info`
`mysql_get_host_info`
`mysql_get_server_info`

1.4.29. `mysql_get_server_info`

- `mysql_get_server_info`

  Get MySQL server info

Description

```
  string mysql_get_server_info(resource link_identifier);
```

Retrieves the MySQL server version.

Parameters

`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns the MySQL server version on success or `FALSE` on failure.

Examples

Example 1.36. `mysql_get_server_info` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
printf("MySQL server version: %s\n", mysql_get_server_info());
?>
```

```
```

The above example will output something similar to:

```
MySQL server version: 4.0.1-alpha
```

See Also

mysql_get_client_info
mysql_get_host_info
mysql_get_proto_info
phpversion

1.4.30. mysql_info

- mysql_info

  Get information about the most recent query

Description

```
string mysql_info(resource link_identifier);
```

Returns detailed information about the last query.

Parameters

link_identifier	The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect is assumed. If no such link is found, it will try to create one as if mysql_connect was called with no arguments. If no connection is found or established, an E_WARNING level error is generated.

Return Values

Returns information about the statement on success, or FALSE on failure. See the example below for which statements provide information, and what the returned value may look like. Statements that are not listed will return FALSE .

Examples

Example 1.37. Relevant MySQL Statements

Statements that return string values. The numbers are only for illustrating purpose; their values will correspond to the query.

```
INSERT INTO ... SELECT ...
String format: Records: 23 Duplicates: 0 Warnings: 0
INSERT INTO ... VALUES (...),(...),(...)...
String format: Records: 37 Duplicates: 0 Warnings: 0
LOAD DATA INFILE ...
String format: Records: 42 Deleted: 0 Skipped: 0 Warnings: 0
ALTER TABLE
String format: Records: 60 Duplicates: 0 Warnings: 0
```

```
UPDATE
String format: Rows matched: 65 Changed: 65 Warnings: 0
```

Notes

Note

`mysql_info` returns a non- `FALSE` value for the INSERT ... VALUES statement only if multiple value lists are specified in the statement.

See Also

`mysql_affected_rows`
`mysql_insert_id`
`mysql_stat`

1.4.31. `mysql_insert_id`

- `mysql_insert_id`

  Get the ID generated in the last query

Description

```
int mysql_insert_id(resource link_identifier);
```

Retrieves the ID generated for an AUTO_INCREMENT column by the previous query (usually INSERT).

Parameters

`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

The ID generated for an AUTO_INCREMENT column by the previous query on success, `0` if the previous query does not generate an AUTO_INCREMENT value, or `FALSE` if no MySQL connection was established.

Examples

Example 1.38. `mysql_insert_id` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
mysql_select_db('mydb');
mysql_query("INSERT INTO mytable (product) values ('kossu')");
printf("Last inserted record has id %d\n", mysql_insert_id());
?>
```

Notes

Caution

`mysql_insert_id` will convert the return type of the native MySQL C API function `mysql_insert_id()` to a type of `long` (named int in PHP). If your AUTO_INCREMENT column has a column type of BIGINT (64 bits) the conversion may result in an incorrect value. Instead, use the internal MySQL SQL function LAST_INSERT_ID() in an SQL query. For more information about PHP's maximum integer values, please see the integer documentation.

Note

Because `mysql_insert_id` acts on the last performed query, be sure to call `mysql_insert_id` immediately after the query that generates the value.

Note

The value of the MySQL SQL function `LAST_INSERT_ID()` always contains the most recently generated AUTO_INCREMENT value, and is not reset between queries.

See Also

`mysql_query`
`mysql_info`

1.4.32. `mysql_list_dbs`

- `mysql_list_dbs`

  List databases available on a MySQL server

Description

```
resource mysql_list_dbs(resource link_identifier);
```

Returns a result pointer containing the databases available from the current mysql daemon.

Parameters

`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns a result pointer resource on success, or `FALSE` on failure. Use the `mysql_tablename` function to traverse this result pointer, or any function for result tables, such as `mysql_fetch_array`.

Examples

Example 1.39. `mysql_list_dbs` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$db_list = mysql_list_dbs($link);
```

```
while ($row = mysql_fetch_object($db_list)) {
     echo $row->Database . "\n";
}
?>
```

The above example will output something similar to:

```
database1
database2
database3
```

Notes

> **Note**
>
> For backward compatibility, the following deprecated alias may be used: `mysql_listdbs`

See Also

`mysql_db_name`
`mysql_select_db`

1.4.33. `mysql_list_fields`

- `mysql_list_fields`

  List MySQL table fields

Description

```
  resource mysql_list_fields(string database_name,
                             string table_name,
                             resource link_identifier);
```

Retrieves information about the given table name.

This function is deprecated. It is preferable to use `mysql_query` to issue an SQL `SHOW COLUMNS FROM table [LIKE 'name']` statement instead.

Parameters

`database_name`	The name of the database that's being queried.
`table_name`	The name of the table that's being queried.
`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

A result pointer resource on success, or `FALSE` on failure.

The returned result can be used with `mysql_field_flags`, `mysql_field_len`, `mysql_field_name` and `mysql_field_type`.

Examples

Example 1.40. Alternate to deprecated `mysql_list_fields`

```
<?php
$result = mysql_query("SHOW COLUMNS FROM sometable");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
if (mysql_num_rows($result) > 0) {
    while ($row = mysql_fetch_assoc($result)) {
        print_r($row);
    }
}
?>
```

The above example will output something similar to:

```

Array
(
    [Field] => id
    [Type] => int(7)
    [Null] =>
    [Key] => PRI
    [Default] =>
    [Extra] => auto_increment
)
Array
(
    [Field] => email
    [Type] => varchar(100)
    [Null] =>
    [Key] =>
    [Default] =>
    [Extra] =>
)
```

Notes

> **Note**
>
> For backward compatibility, the following deprecated alias may be used: `mysql_listfields`

See Also

`mysql_field_flags`
`mysql_info`

1.4.34. `mysql_list_processes`

- `mysql_list_processes`

  List MySQL processes

Description

```
resource mysql_list_processes(resource link_identifier);
```

Retrieves the current MySQL server threads.

Parameters

`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

A result pointer resource on success or `FALSE` on failure.

Examples

Example 1.41. `mysql_list_processes` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$result = mysql_list_processes($link);
while ($row = mysql_fetch_assoc($result)){
    printf("%s %s %s %s %s\n", $row["Id"], $row["Host"], $row["db"],
        $row["Command"], $row["Time"]);
}
mysql_free_result($result);
?>
```

The above example will output something similar to:

```
1 localhost test Processlist 0
4 localhost mysql sleep 5
```

See Also

`mysql_thread_id`
`mysql_stat`

1.4.35. `mysql_list_tables`

- `mysql_list_tables`

  List tables in a MySQL database

Description

```
resource mysql_list_tables(string database,
                           resource link_identifier);
```

Retrieves a list of table names from a MySQL database.

This function is deprecated. It is preferable to use `mysql_query` to issue an SQL `SHOW TABLES [FROM db_name] [LIKE 'pattern']` statement instead.

Parameters

`database`	The name of the database
`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

A result pointer resource on success or `FALSE` on failure.

Use the `mysql_tablename` function to traverse this result pointer, or any function for result tables, such as `mysql_fetch_array`.

Changelog

Version	Description
4.3.7	This function became deprecated.

Examples

Example 1.42. `mysql_list_tables` alternative example

```
<?php
$dbname = 'mysql_dbname';
if (!mysql_connect('mysql_host', 'mysql_user', 'mysql_password')) {
    echo 'Could not connect to mysql';
    exit;
}
$sql = "SHOW TABLES FROM $dbname";
$result = mysql_query($sql);
if (!$result) {
    echo "DB Error, could not list tables\n";
    echo 'MySQL Error: ' . mysql_error();
    exit;
}
while ($row = mysql_fetch_row($result)) {
    echo "Table: {$row[0]}\n";
}
mysql_free_result($result);
?>
```

Notes

> **Note**
>
> For backward compatibility, the following deprecated alias may be used: `mysql_listtables`

See Also

`mysql_list_dbs`

`mysql_tablename`

1.4.36. `mysql_num_fields`

- `mysql_num_fields`

  Get number of fields in result

Description

```
int mysql_num_fields(resource result);
```

Retrieves the number of fields from a query.

Parameters

`result`	The result resource that is being evaluated. This result comes from a call to `mysql_query`.

Return Values

Returns the number of fields in the result set resource on success or `FALSE` on failure.

Examples

Example 1.43. A `mysql_num_fields` example

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
/* returns 2 because id,email === two fields */
echo mysql_num_fields($result);
?>
```

Notes

Note

For backward compatibility, the following deprecated alias may be used: `mysql_numfields`

See Also

`mysql_select_db`
`mysql_query`
`mysql_fetch_field`
`mysql_num_rows`

1.4.37. `mysql_num_rows`

- mysql_num_rows

  Get number of rows in result

Description

```
int mysql_num_rows(resource result);
```

Retrieves the number of rows from a result set. This command is only valid for statements like SELECT or SHOW that return an actual result set. To retrieve the number of rows affected by a INSERT, UPDATE, REPLACE or DELETE query, use mysql_affected_rows.

Parameters

result	The result resource that is being evaluated. This result comes from a call to mysql_query.

Return Values

The number of rows in a result set on success or FALSE on failure.

Examples

Example 1.44. mysql_num_rows example

```
<?php
$link = mysql_connect("localhost", "mysql_user", "mysql_password");
mysql_select_db("database", $link);
$result = mysql_query("SELECT * FROM table1", $link);
$num_rows = mysql_num_rows($result);
echo "$num_rows Rows\n";
?>
```

Notes

Note

If you use mysql_unbuffered_query, mysql_num_rows will not return the correct value until all the rows in the result set have been retrieved.

Note

For backward compatibility, the following deprecated alias may be used: mysql_numrows

See Also

mysql_affected_rows
mysql_connect
mysql_data_seek
mysql_select_db
mysql_query

1.4.38. mysql_pconnect

- `mysql_pconnect`

  Open a persistent connection to a MySQL server

Description

```
resource mysql_pconnect(string server= =ini_get("mysql.default_host"),
                        string username= =ini_get("mysql.default_user"),
                        string password= =ini_get("mysql.default_password"),
                        int client_flags);
```

Establishes a persistent connection to a MySQL server.

`mysql_pconnect` acts very much like `mysql_connect` with two major differences.

First, when connecting, the function would first try to find a (persistent) link that's already open with the same host, username and password. If one is found, an identifier for it will be returned instead of opening a new connection.

Second, the connection to the SQL server will not be closed when the execution of the script ends. Instead, the link will remain open for future use (`mysql_close` will not close links established by `mysql_pconnect`).

This type of link is therefore called 'persistent'.

Parameters

`server`	The MySQL server. It can also include a port number. e.g. "hostname:port" or a path to a local socket e.g. ":/path/to/socket" for the localhost. If the PHP directive mysql.default_host is undefined (default), then the default value is 'localhost:3306'
`username`	The username. Default value is the name of the user that owns the server process.
`password`	The password. Default value is an empty password.
`client_flags`	The *`client_flags`* parameter can be a combination of the following constants: 128 (enable `LOAD DATA LOCAL` handling), `MYSQL_CLIENT_SSL` , `MYSQL_CLIENT_COMPRESS` , `MYSQL_CLIENT_IGNORE_SPACE` or `MYSQL_CLIENT_INTERACTIVE` .

Return Values

Returns a MySQL persistent link identifier on success, or `FALSE` on failure.

Changelog

Version	Description
4.3.0	Added the *`client_flags`* parameter.

Notes

Note

Note, that these kind of links only work if you are using a module version of PHP. See the Persistent Database Connections section for more information.

Warning

Using persistent connections can require a bit of tuning of your Apache and MySQL configurations to ensure that you do not exceed the number of connections allowed by MySQL.

Note

You can suppress the error message on failure by prepending a @ to the function name.

See Also

mysql_connect
Persistent Database Connections

1.4.39. mysql_ping

- mysql_ping

  Ping a server connection or reconnect if there is no connection

Description

```
bool mysql_ping(resource link_identifier);
```

Checks whether or not the connection to the server is working. If it has gone down, an automatic reconnection is attempted. This function can be used by scripts that remain idle for a long while, to check whether or not the server has closed the connection and reconnect if necessary.

Note

Since MySQL 5.0.13, automatic reconnection feature is disabled.

Parameters

link_identifier	The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect is assumed. If no such link is found, it will try to create one as if mysql_connect was called with no arguments. If no connection is found or established, an E_WARNING level error is generated.

Return Values

Returns TRUE if the connection to the server MySQL server is working, otherwise FALSE .

Examples

Example 1.45. A mysql_ping example

```
<?php
set_time_limit(0);
$conn = mysql_connect('localhost', 'mysqluser', 'mypass');
$db   = mysql_select_db('mydb');
/* Assuming this query will take a long time */
$result = mysql_query($sql);
if (!$result) {
    echo 'Query #1 failed, exiting.';
    exit;
}
/* Make sure the connection is still alive, if not, try to reconnect */
if (!mysql_ping($conn)) {
    echo 'Lost connection, exiting after query #1';
    exit;
}
mysql_free_result($result);
/* So the connection is still alive, let's run another query */
```

```
$result2 = mysql_query($sql2);
?>
```

See Also

mysql_thread_id
mysql_list_processes

1.4.40. mysql_query

- mysql_query

  Send a MySQL query

Description

```
 resource mysql_query(string query,
                      resource link_identifier);
```

mysql_query sends a unique query (multiple queries are not supported) to the currently active database on the server that's associated with the specified *link_identifier*.

Parameters

query	An SQL query The query string should not end with a semicolon. Data inside the query should be properly escaped.
link_identifier	The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect is assumed. If no such link is found, it will try to create one as if mysql_connect was called with no arguments. If no connection is found or established, an E_WARNING level error is generated.

Return Values

For SELECT, SHOW, DESCRIBE, EXPLAIN and other statements returning resultset, mysql_query returns a resource on success, or FALSE on error.

For other type of SQL statements, INSERT, UPDATE, DELETE, DROP, etc, mysql_query returns TRUE on success or FALSE on error.

The returned result resource should be passed to mysql_fetch_array, and other functions for dealing with result tables, to access the returned data.

Use mysql_num_rows to find out how many rows were returned for a SELECT statement or mysql_affected_rows to find out how many rows were affected by a DELETE, INSERT, REPLACE, or UPDATE statement.

mysql_query will also fail and return FALSE if the user does not have permission to access the table(s) referenced by the query.

Examples

Example 1.46. Invalid Query

The following query is syntactically invalid, so `mysql_query` fails and returns `FALSE` .

```
<?php
$result = mysql_query('SELECT * WHERE 1=1');
if (!$result) {
    die('Invalid query: ' . mysql_error());
}
?>
```

Example 1.47. Valid Query

The following query is valid, so `mysql_query` returns a resource.

```
<?php
// This could be supplied by a user, for example
$firstname = 'fred';
$lastname  = 'fox';
// Formulate Query
// This is the best way to perform an SQL query
// For more examples, see mysql_real_escape_string()
$query = sprintf("SELECT firstname, lastname, address, age FROM friends WHERE firstname='%s' AND lastname='%s'",
    mysql_real_escape_string($firstname),
    mysql_real_escape_string($lastname));
// Perform Query
$result = mysql_query($query);
// Check result
// This shows the actual query sent to MySQL, and the error. Useful for debugging.
if (!$result) {
    $message  = 'Invalid query: ' . mysql_error() . "\n";
    $message .= 'Whole query: ' . $query;
    die($message);
}
// Use result
// Attempting to print $result won't allow access to information in the resource
// One of the mysql result functions must be used
// See also mysql_result(), mysql_fetch_array(), mysql_fetch_row(), etc.
while ($row = mysql_fetch_assoc($result)) {
    echo $row['firstname'];
    echo $row['lastname'];
    echo $row['address'];
    echo $row['age'];
}
// Free the resources associated with the result set
// This is done automatically at the end of the script
mysql_free_result($result);
?>
```

See Also

`mysql_connect`
`mysql_error`
`mysql_real_escape_string`
`mysql_result`
`mysql_fetch_assoc`
`mysql_unbuffered_query`

1.4.41. `mysql_real_escape_string`

- `mysql_real_escape_string`

  Escapes special characters in a string for use in an SQL statement

Description

```
string mysql_real_escape_string(string unescaped_string,
                                resource link_identifier);
```

Escapes special characters in the *`unescaped_string`*, taking into account the current character set of the connection so that it is safe to place it in a `mysql_query`. If binary data is to be inserted, this function must be used.

`mysql_real_escape_string` calls MySQL's library function mysql_real_escape_string, which prepends backslashes to the following characters: `\x00`, `\n`, `\r`, `\`, `'`, `"` and `\x1a`.

This function must always (with few exceptions) be used to make data safe before sending a query to MySQL.

Parameters

`unescaped_string`	The string that is to be escaped.
`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns the escaped string, or `FALSE` on error.

Examples

Example 1.48. Simple `mysql_real_escape_string` example

```
<?php
// Connect
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
    OR die(mysql_error());
// Query
$query = sprintf("SELECT * FROM users WHERE user='%s' AND password='%s'",
            mysql_real_escape_string($user),
            mysql_real_escape_string($password));
?>
```

Example 1.49. An example SQL Injection Attack

```
<?php
// Query database to check if there are any matching users
$query = "SELECT * FROM users WHERE user='{$_POST['username']}' AND password='{$_POST['password']}'";
mysql_query($query);
// We didn't check $_POST['password'], it could be anything the user wanted! For example:
$_POST['username'] = 'aidan';
$_POST['password'] = "' OR ''='";
// This means the query sent to MySQL would be:
echo $query;
?>
```

The query sent to MySQL:

```
SELECT * FROM users WHERE user='aidan' AND password='' OR ''=''
```

This would allow anyone to log in without a valid password.

Notes

> **Note**
>
> A MySQL connection is required before using `mysql_real_escape_string` otherwise an error of level `E_WARNING` is generated, and `FALSE` is returned. If *`link_identifier`* isn't defined, the last MySQL connection is used.

> **Note**
>
> If magic_quotes_gpc is enabled, first apply `stripslashes` to the data. Using this function on data which has already been escaped will escape the data twice.

> **Note**
>
> If this function is not used to escape data, the query is vulnerable to SQL Injection Attacks.

> **Note**
>
> `mysql_real_escape_string` does not escape `%` and `_`. These are wildcards in MySQL if combined with `LIKE`, `GRANT`, or `REVOKE`.

See Also

`mysql_client_encoding`
`addslashes`
`stripslashes`
The magic_quotes_gpc directive
The magic_quotes_runtime directive

1.4.42. `mysql_result`

- `mysql_result`

  Get result data

Description

```
  string mysql_result(resource result,
                      int row,
                      mixed field= =0);
```

Retrieves the contents of one cell from a MySQL result set.

When working on large result sets, you should consider using one of the functions that fetch an entire row (specified below). As these functions return the contents of multiple cells in one function call, they're MUCH quicker than `mysql_result`. Also, note that specifying a numeric offset for the field argument is much quicker than specifying a fieldname or tablename.fieldname argument.

Parameters

`result`	The result resource that is being evaluated. This result comes from a call to `mysql_query`.

row	The row number from the result that's being retrieved. Row numbers start at `0`.
field	The name or offset of the field being retrieved. It can be the field's offset, the field's name, or the field's table dot field name (tablename.fieldname). If the column name has been aliased ('select foo as bar from...'), use the alias instead of the column name. If undefined, the first field is retrieved.

Return Values

The contents of one cell from a MySQL result set on success, or `FALSE` on failure.

Examples

Example 1.50. `mysql_result` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
if (!mysql_select_db('database_name')) {
    die('Could not select database: ' . mysql_error());
}
$result = mysql_query('SELECT name FROM work.employee');
if (!$result) {
    die('Could not query:' . mysql_error());
}
echo mysql_result($result, 2); // outputs third employee's name
mysql_close($link);
?>
```

Notes

> **Note**
>
> Calls to `mysql_result` should not be mixed with calls to other functions that deal with the result set.

See Also

`mysql_fetch_row`
`mysql_fetch_array`
`mysql_fetch_assoc`
`mysql_fetch_object`

1.4.43. `mysql_select_db`

- `mysql_select_db`

  Select a MySQL database

Description

```
  bool mysql_select_db(string database_name,
                       resource link_identifier);
```

Sets the current active database on the server that's associated with the specified link identifier. Every subsequent call to mysql_query will be made on the active database.

Parameters

database_name	The name of the database that is to be selected.
link_identifier	The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect is assumed. If no such link is found, it will try to create one as if mysql_connect was called with no arguments. If no connection is found or established, an E_WARNING level error is generated.

Return Values

Returns TRUE on success or FALSE on failure.

Examples

Example 1.51. mysql_select_db example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Not connected : ' . mysql_error());
}
// make foo the current db
$db_selected = mysql_select_db('foo', $link);
if (!$db_selected) {
    die ('Can\'t use foo : ' . mysql_error());
}
?>
```

Notes

> **Note**
>
> For backward compatibility, the following deprecated alias may be used: mysql_selectdb

See Also

mysql_connect
mysql_pconnect
mysql_query

1.4.44. mysql_set_charset

- mysql_set_charset

  Sets the client character set

Description

```
  bool mysql_set_charset(string charset,
                         resource link_identifier);
```

Sets the default character set for the current connection.

Parameters

`charset`	A valid character set name.
`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

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

Notes

Note

This function requires MySQL 5.0.7 or later.

Note

This is the preferred way to change the charset. Using `mysql_query` to execute `SET NAMES ..` is not recommended.

See Also

`mysql_client_encoding`
List of character sets that MySQL supports

1.4.45. `mysql_stat`

- `mysql_stat`

  Get current system status

Description

```
string mysql_stat(resource link_identifier);
```

`mysql_stat` returns the current server status.

Parameters

`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns a string with the status for uptime, threads, queries, open tables, flush tables and queries per second. For a complete list of other status variables, you have to use the `SHOW STATUS` SQL command. If *`link_identifier`* is invalid, `NULL` is returned.

Examples

Example 1.52. `mysql_stat` example

```
<?php
$link   = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$status = explode('  ', mysql_stat($link));
print_r($status);
?>
```

The above example will output something similar to:

```
Array
(
    [0] => Uptime: 5380
    [1] => Threads: 2
    [2] => Questions: 1321299
    [3] => Slow queries: 0
    [4] => Opens: 26
    [5] => Flush tables: 1
    [6] => Open tables: 17
    [7] => Queries per second avg: 245.595
)
```

Example 1.53. Alternative `mysql_stat` example

```
<?php
$link   = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$result = mysql_query('SHOW STATUS', $link);
while ($row = mysql_fetch_assoc($result)) {
    echo $row['Variable_name'] . ' = ' . $row['Value'] . "\n";
}
?>
```

The above example will output something similar to:

```
back_log = 50
basedir = /usr/local/
bdb_cache_size = 8388600
bdb_log_buffer_size = 32768
bdb_home = /var/db/mysql/
bdb_max_lock = 10000
bdb_logdir =
bdb_shared_data = OFF
bdb_tmpdir = /var/tmp/
...
```

See Also

`mysql_get_server_info`
`mysql_list_processes`

1.4.46. `mysql_tablename`

- `mysql_tablename`

  Get table name of field

Description

```
string mysql_tablename(resource result,
                       int i);
```

Retrieves the table name from a *`result`*.

This function is deprecated. It is preferable to use `mysql_query` to issue an SQL `SHOW TABLES [FROM db_name] [LIKE 'pattern']` statement instead.

Parameters

`result`	A result pointer resource that's returned from `mysql_list_tables`.
`i`	The integer index (row/table number)

Return Values

The name of the table on success or `FALSE` on failure.

Use the `mysql_tablename` function to traverse this result pointer, or any function for result tables, such as `mysql_fetch_array`.

Examples

Example 1.54. `mysql_tablename` example

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password");
$result = mysql_list_tables("mydb");
$num_rows = mysql_num_rows($result);
for ($i = 0; $i < $num_rows; $i++) {
    echo "Table: ", mysql_tablename($result, $i), "\n";
}
mysql_free_result($result);
?>
```

Notes

> **Note**
>
> The `mysql_num_rows` function may be used to determine the number of tables in the result pointer.

See Also

`mysql_list_tables`
`mysql_field_table`
`mysql_db_name`

1.4.47. `mysql_thread_id`

- `mysql_thread_id`

  Return the current thread ID

Description

```
int mysql_thread_id(resource link_identifier);
```

Retrieves the current thread ID. If the connection is lost, and a reconnect with `mysql_ping` is executed, the thread ID will change. This means only retrieve the thread ID when needed.

Parameters

`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

The thread ID on success or `FALSE` on failure.

Examples

Example 1.55. `mysql_thread_id` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$thread_id = mysql_thread_id($link);
if ($thread_id){
    printf("current thread id is %d\n", $thread_id);
}
?>
```

The above example will output something similar to:

```
current thread id is 73
```

See Also

`mysql_ping`
`mysql_list_processes`

1.4.48. `mysql_unbuffered_query`

- `mysql_unbuffered_query`

  Send an SQL query to MySQL without fetching and buffering the result rows.

Description

```
resource mysql_unbuffered_query(string query,
                                resource link_identifier);
```

`mysql_unbuffered_query` sends the SQL query *`query`* to MySQL without automatically fetching and buffering the result rows as `mysql_query` does. This saves a considerable amount of memory with SQL queries that produce large result sets, and you can start working on the result set immediately after the first row has been retrieved as you don't have to wait until the complete SQL query has been performed. To use `mysql_unbuffered_query` while multiple database connections are open, you must specify the optional parameter *`link_identifier`* to identify which connection you want to use.

Parameters

`query`	The SQL query to execute. Data inside the query should be properly escaped.
`link_identifier`	The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` was called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

For SELECT, SHOW, DESCRIBE or EXPLAIN statements, `mysql_unbuffered_query` returns a resource on success, or `FALSE` on error.

For other type of SQL statements, UPDATE, DELETE, DROP, etc, `mysql_unbuffered_query` returns `TRUE` on success or `FALSE` on error.

Notes

Note

The benefits of `mysql_unbuffered_query` come at a cost: you cannot use `mysql_num_rows` and `mysql_data_seek` on a result set returned from `mysql_unbuffered_query`. You also have to fetch all result rows from an unbuffered SQL query before you can send a new SQL query to MySQL.

See Also

`mysql_query`

Chapter 2. MySQL Improved Extension (`Mysqli`)

The `mysqli` extension allows you to access the functionality provided by MySQL 4.1 and above. More information about the MySQL Database server can be found at http://www.mysql.com/

An overview of software available for using MySQL from PHP can be found at Section 2.2, “Overview”

Documentation for MySQL can be found at http://dev.mysql.com/doc/.

Parts of this documentation included from MySQL manual with permissions of Oracle Corporation.

2.1. Examples

All examples in the `mysqli` documentation use the world database. The world database can be found at http://downloads.mysql.com/docs/world.sql.gz

2.2. Overview

This section provides an introduction to the options available to you when developing a PHP application that needs to interact with a MySQL database.

What is an API?

An Application Programming Interface, or API, defines the classes, methods, functions and variables that your application will need to call in order to carry out its desired task. In the case of PHP applications that need to communicate with databases the necessary APIs are usually exposed via PHP extensions.

APIs can be procedural or object-oriented. With a procedural API you call functions to carry out tasks, with the object-oriented API you instantiate classes and then call methods on the resulting objects. Of the two the latter is usually the preferred interface, as it is more modern and leads to better organised code.

When writing PHP applications that need to connect to the MySQL server there are several API options available. This document discusses what is available and how to select the best solution for your application.

What is a Connector?

In the MySQL documentation, the term *connector* refers to a piece of software that allows your application to connect to the MySQL database server. MySQL provides connectors for a variety of languages, including PHP.

If your PHP application needs to communicate with a database server you will need to write PHP code to perform such activities as connecting to the database server, querying the database and other database-related functions. Software is required to provide the API that your PHP application will use, and also handle the communication between your application and the database server, possibly using other intermediate libraries where necessary. This software is known generically as a connector, as it allows your application to *connect* to a database server.

What is a Driver?

A driver is a piece of software designed to communicate with a specific type of database server. The driver may also call a library, such as the MySQL Client Library or the MySQL Native Driver. These libraries implement the low-level protocol used to communicate with the MySQL database server.

By way of an example, the PHP Data Objects (PDO) database abstraction layer may use one of several database-specific drivers. One of the drivers it has available is the PDO MYSQL driver, which allows it to interface with the MySQL server.

Sometimes people use the terms connector and driver interchangeably, this can be confusing. In the MySQL-related documentation the term “driver” is reserved for software that provides the database-specific part of a connector package.

What is an Extension?

In the PHP documentation you will come across another term - *extension*. The PHP code consists of a core, with optional extensions to the core functionality. PHP's MySQL-related extensions, such as the `mysqli` extension, and the `mysql` extension, are implemented using the PHP extension framework.

An extension typically exposes an API to the PHP programmer, to allow its facilities to be used programmatically. However, some extensions which use the PHP extension framework do not expose an API to the PHP programmer.

The PDO MySQL driver extension, for example, does not expose an API to the PHP programmer, but provides an interface to the PDO layer above it.

The terms API and extension should not be taken to mean the same thing, as an extension may not necessarily expose an API to the programmer.

What are the main PHP API offerings for using MySQL?

There are three main API options when considering connecting to a MySQL database server:

- PHP's MySQL Extension
- PHP's mysqli Extension
- PHP Data Objects (PDO)

Each has its own advantages and disadvantages. The following discussion aims to give a brief introduction to the key aspects of each API.

What is PHP's MySQL Extension?

This is the original extension designed to allow you to develop PHP applications that interact with a MySQL database. The `mysql` extension provides a procedural interface and is intended for use only with MySQL versions older than 4.1.3. This extension can be used with versions of MySQL 4.1.3 or newer, but not all of the latest MySQL server features will be available.

Note

If you are using MySQL versions 4.1.3 or later it is *strongly* recommended that you use the `mysqli` extension instead.

The `mysql` extension source code is located in the PHP extension directory `ext/mysql`.

For further information on the `mysql` extension, see Chapter 1, *MySQL Extension (mysql)*.

What is PHP's mysqli Extension?

The `mysqli` extension, or as it is sometimes known, the MySQL *improved* extension, was developed to take advantage of new features found in MySQL systems versions 4.1.3 and newer. The `mysqli` extension is included with PHP versions 5 and later.

The `mysqli` extension has a number of benefits, the key enhancements over the `mysql` extension being:

- Object-oriented interface
- Support for Prepared Statements
- Support for Multiple Statements
- Support for Transactions
- Enhanced debugging capabilities
- Embedded server support

Note

> If you are using MySQL versions 4.1.3 or later it is *strongly* recommended that you use this extension.

As well as the object-oriented interface the extension also provides a procedural interface.

The `mysqli` extension is built using the PHP extension framework, its source code is located in the directory `ext/mysqli`.

For further information on the `mysqli` extension, see Chapter 2, *MySQL Improved Extension (`Mysqli`)*.

What is PDO?

PHP Data Objects, or PDO, is a database abstraction layer specifically for PHP applications. PDO provides a consistent API for your PHP application regardless of the type of database server your application will connect to. In theory, if you are using the PDO API, you could switch the database server you used, from say Firebird to MySQL, and only need to make minor changes to your PHP code.

Other examples of database abstraction layers include JDBC for Java applications and DBI for Perl.

While PDO has its advantages, such as a clean, simple, portable API, its main disadvantage is that it doesn't allow you to use all of the advanced features that are available in the latest versions of MySQL server. For example, PDO does not allow you to use MySQL's support for Multiple Statements.

PDO is implemented using the PHP extension framework, its source code is located in the directory `ext/pdo`.

For further information on PDO, see the Chapter 4, *MySQL Functions (PDO_MYSQL)*.

What is the PDO MYSQL driver?

The PDO MYSQL driver is not an API as such, at least from the PHP programmer's perspective. In fact the PDO MYSQL driver sits in the layer below PDO itself and provides MySQL-specific functionality. The programmer still calls the PDO API, but PDO uses the PDO MYSQL driver to carry out communication with the MySQL server.

The PDO MYSQL driver is one of several available PDO drivers. Other PDO drivers available include those for the Firebird and PostgreSQL database servers.

The PDO MYSQL driver is implemented using the PHP extension framework. Its source code is located in the directory `ext/pdo_mysql`. It does not expose an API to the PHP programmer.

For further information on the PDO MYSQL driver, see Chapter 4, *MySQL Functions (PDO_MYSQL)*.

What is PHP's MySQL Native Driver?

In order to communicate with the MySQL database server the `mysql` extension, `mysqli` and the PDO MYSQL driver each use a low-level library that implements the required protocol. In the past, the only available library was the MySQL Client Library, otherwise known as `libmysql`.

However, the interface presented by `libmysql` was not optimized for communication with PHP applications, as `libmysql` was originally designed with C applications in mind. For this reason the MySQL Native Driver, `mysqlnd`, was developed as an alternative to `libmysql` for PHP applications.

The `mysql` extension, the `mysqli` extension and the PDO MySQL driver can each be individually configured to use either `libmysql` or `mysqlnd`. As `mysqlnd` is designed specifically to be utilised in the PHP system it has numerous memory and speed enhancements over `libmysql`. You are strongly encouraged to take advantage of these improvements.

> **Note**
>
> The MySQL Native Driver can only be used with MySQL server versions 4.1.3 and later.

The MySQL Native Driver is implemented using the PHP extension framework. The source code is located in `ext/mysqlnd`. It does not expose an API to the PHP programmer.

Comparison of Features

The following table compares the functionality of the three main methods of connecting to MySQL from PHP:

	PHP's mysqli Extension	PDO (Using PDO MySQL Driver and MySQL Native Driver)	PHP's MySQL Extension
PHP version introduced	5.0	5.0	Prior to 3.0
Included with PHP 5.x	yes	yes	Yes
MySQL development status	Active development	Active development as of PHP 5.3	Maintenance only
Recommended by MySQL for new projects	Yes - preferred option	Yes	No
API supports Charsets	Yes	Yes	No
API supports server-side Prepared Statements	Yes	Yes	No
API supports client-side Prepared Statements	No	Yes	No
API supports Stored Procedures	Yes	Yes	No
API supports Multiple Statements	Yes	Most	No
Supports all MySQL 4.1+ functionality	Yes	Most	No

2.3. Installing/Configuring

2.3.1. Requirements

In order to have these functions available, you must compile PHP with support for the mysqli extension.

Note

The mysqli extension is designed to work with MySQL version 4.1.13 or newer, or 5.0.7 or newer. For previous versions, please see the MySQL extension documentation.

2.3.2. Installation

The `mysqli` extension was introduced with PHP version 5.0.0. The MySQL Native Driver was included in PHP version 5.3.0.

2.3.2.1. Installation on Linux

The common Unix distributions include binary versions of PHP that can be installed. Although these binary versions are typically built with support for MySQL extensions enabled, the extension libraries themselves may need to be installed using an additional package. Check the package manager than comes with your chosen distribution for availability.

Unless your Unix distribution comes with a binary package of PHP with the `mysqli` extension available, you will need to build PHP from source code. Building PHP from source allows you to specify the MySQL extensions you want to use, as well as your choice of client library for each extension.

2.3.2.1.1. PHP 5.0, 5.1, 5.2

If building from source code, to ensure that the mysqli extension for PHP is enabled, you will need to configure the PHP source code to use mysqli. This is achieved by running the configure script with the option --with-mysqli=mysql_config_path/mysql_config, prior to building PHP. This will enable mysqli and it will use the MySQL Client Library (libmysql) to communicate with the MySQL Server.

The mysql_config_path represents the location of the mysql_config program that comes with MySQL Server.

2.3.2.1.2. PHP 5.3.0+

With versions of PHP 5.3.0 and newer, mysqli uses MySQL Native Driver by default. This gives a number of benefits over libmysql.

This is the recommended option, as using the MySQL Native Driver results in improved performance and gives access to features not available when using the MySQL Client Library. Refer to What is PHP's MySQL Native Driver? for a brief overview of the advantages of MySQL Native Driver.

To use MySQL Native Driver with mysqli you need to configure the PHP source code using the --with-mysqli=mysqlnd option, prior to building PHP.

Note that it is possible to freely mix MySQL extensions and client libraries. For example, it is possible to enable the MySQL extension to use the MySQL Client Library (libmysql), while configuring the mysqli extension to use the MySQL Native Driver. However, all permutations of extension and client library are possible.

The following example builds the MySQL extension to use the MySQL Client Library, and the mysqli and PDO MYSQL extensions to use the MySQL Native Driver:

```
./configure --with-mysql=/usr/bin/mysql_config  \
--with-mysqli=mysqlnd \
--with-pdo-mysql=mysqlnd
[other options]
```

2.3.2.2. Installation on Windows Systems

On Windows, PHP is most commonly installed using the binary installer.

2.3.2.2.1. PHP 5.0, 5.1, 5.2

Once PHP has been installed, some configuration is required to enable mysqli and specify the client library you want it to use.

The mysqli extension is not enabled by default, so the php_mysqli.dll DLL must be enabled inside of php.ini. In order to do this you need to find the php.ini file (typically located in c:\php), and make sure you remove the comment (semi-colon) from the start of the line extension=php_mysqli.dll, in the section marked [PHP_MYSQLI].

Also, if you want to use the MySQL Client Library with mysqli, you need to make sure PHP can access the client library file. The MySQL Client Library is included as a file named libmysql.dll in the Windows PHP distribution. This file needs to be available in the Windows system's PATH environment variable, so that it can be successfully loaded. See the FAQ titled "How do I add my PHP directory to the PATH on Windows" for information on how to do this. Copying libmysql.dll to the Windows system directory (typically c:\Windows\system) also works, as the system directory is by default in the system's PATH. However, this practice is strongly discouraged.

As with enabling any PHP extension (such as php_mysqli.dll), the PHP directive extension_dir should be set to the directory where the PHP extensions are located. See also the Manual Windows Installation Instructions. An example extension_dir value for PHP 5 is c:\php\ext.

Note

If when starting the web server an error similar to the following occurs: "Unable to load dynamic library './php_mysqli.dll'", this is because php_mysqli.dll and/or libmysql.dll cannot be found by the system.

2.3.2.2.2. PHP 5.3.0+

On Windows, for PHP versions 5.3 and newer, the `mysqli` extension is enabled and uses the MySQL Native Driver by default. This means you don't need to worry about configuring access to `libmysql.dll`.

2.3.3. Runtime Configuration

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

Table 2.1. MySQLi Configuration Options

Name	Default	Changeable	Changelog
mysqli.allow_persistent	"1"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqli.max_persistent	"-1"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqli.max_links	"-1"	PHP_INI_SYSTEM	Available since PHP 5.0.0.
mysqli.default_port	"3306"	PHP_INI_ALL	Available since PHP 5.0.0.
mysqli.default_socket	NULL	PHP_INI_ALL	Available since PHP 5.0.0.
mysqli.default_host	NULL	PHP_INI_ALL	Available since PHP 5.0.0.
mysqli.default_user	NULL	PHP_INI_ALL	Available since PHP 5.0.0.
mysqli.default_pw	NULL	PHP_INI_ALL	Available since PHP 5.0.0.
mysqli.reconnect	"0"	PHP_INI_SYSTEM	Available since PHP 4.3.5.
mysqli.allow_local_infile	"1"	PHP_INI_SYSTEM	Available since PHP 5.2.4.
mysqli.cache_size	"2000"	PHP_INI_SYSTEM	Available since PHP 5.3.0.

For further details and definitions of the preceding PHP_INI_* constants, see the chapter on configuration changes.

Here's a short explanation of the configuration directives.

`mysqli.allow_persistent` integer	Enable the ability to create persistent connections using `mysqli_connect`.
`mysqli.max_persistent` integer	Maximum of persistent connections that can be made. Set to 0 for unlimited.
`mysqli.max_links` integer	The maximum number of MySQL connections per process.
`mysqli.default_port` integer	The default TCP port number to use when connecting to the database server if no other port is specified. If no default is specified, the port will be obtained from the `MYSQL_TCP_PORT` environment variable, the `mysql-tcp` entry in `/etc/services` or the compile-time `MYSQL_PORT` constant, in that order. Win32 will only use the `MYSQL_PORT` constant.
`mysqli.default_socket` string	The default socket name to use when connecting to a local database server if no other socket name is specified.
`mysqli.default_host` string	The default server host to use when connecting to the database server if no other host is specified. Doesn't apply in safe mode.
`mysqli.default_user` string	The default user name to use when connecting to the database server if no other name is specified. Doesn't apply in safe mode.
`mysqli.default_pw` string	The default password to use when connecting to the database server if no other password is specified. Doesn't apply in safe mode.

mysqli.reconnect integer	Automatically reconnect if the connection was lost.
mysqli.allow_local_infile integer	
mysqli.cache_size integer	Available only with mysqlnd.

Users cannot set `MYSQL_OPT_READ_TIMEOUT` through an API call or runtime configuration setting. Note that if it were possible there would be differences between how `libmysql` and streams would interpret the value of `MYSQL_OPT_READ_TIMEOUT`.

2.3.4. Resource Types

This extension has no resource types defined.

2.4. The mysqli Extension and Persistent Connections

Persistent connection support was introduced in PHP 5.3 for the `mysqli` extension. Support was already present in PDO MYSQL and ext/mysql. The idea behind persistent connections is that a connection between a client process and a database can be reused by a client process, rather than being created and destroyed multiple times. This reduces the overhead of creating fresh connections every time one is required, as unused connections are cached and ready to be reused.

Unlike the mysql extension, mysqli does not provide a separate function for opening persistent connections. To open a persistent connection you must prepend `p:` to the hostname when connecting.

The problem with persistent connections is that they can be left in unpredictable states by clients. For example, a table lock might be activated before a client terminates unexpectedly. A new client process reusing this persistent connection will get the connection "as is". Any cleanup would need to be done by the new client process before it could make good use of the persistent connection, increasing the burden on the programmer.

The persistent connection of the `mysqli` extension however provides built-in cleanup handling code. The cleanup carried out by `mysqli` includes:

- Rollback active transactions
- Close and drop temporary tables
- Unlock tables
- Reset session variables
- Close prepared statements (always happens with PHP)
- Close handler
- Release locks acquired with `GET_LOCK`

This ensures that persistent connections are in a clean state on return from the connection pool, before the client process uses them.

The `mysqli` extension does this cleanup by automatically calling the C-API function `mysql_change_user()`.

The automatic cleanup feature has advantages and disadvantages though. The advantage is that the programmer no longer needs to worry about adding cleanup code, as it is called automatically. However, the disadvantage is that the code could *potentially* be a little slower, as the code to perform the cleanup needs to run each time a connection is returned from the connection pool.

It is possible to switch off the automatic cleanup code, by compiling PHP with `MYSQLI_NO_CHANGE_USER_ON_PCONNECT` defined.

Note

The `mysqli` extension supports persistent connections when using either MySQL Native Driver or MySQL Client Library.

2.5. Predefined Constants

MYSQLI_READ_DEFAULT_GROUP	Read options from the named group from my.cnf or the file specified with MYSQLI_READ_DEFAULT_FILE
MYSQLI_READ_DEFAULT_FILE	Read options from the named option file instead of from my.cnf
MYSQLI_OPT_CONNECT_TIMEOUT	Connect timeout in seconds
MYSQLI_OPT_LOCAL_INFILE	Enables command LOAD LOCAL INFILE
MYSQLI_INIT_COMMAND	Command to execute when connecting to MySQL server. Will automatically be re-executed when reconnecting.
MYSQLI_CLIENT_SSL	Use SSL (encrypted protocol). This option should not be set by application programs; it is set internally in the MySQL client library
MYSQLI_CLIENT_COMPRESS	Use compression protocol
MYSQLI_CLIENT_INTERACTIVE	Allow interactive_timeout seconds (instead of wait_timeout seconds) of inactivity before closing the connection. The client's session wait_timeout variable will be set to the value of the session interactive_timeout variable.
MYSQLI_CLIENT_IGNORE_SPACE	Allow spaces after function names. Makes all functions names reserved words.
MYSQLI_CLIENT_NO_SCHEMA	Don't allow the db_name.tbl_name.col_name syntax.
MYSQLI_CLIENT_MULTI_QUERIES	Allows multiple semicolon-delimited queries in a single mysqli_query call.
MYSQLI_STORE_RESULT	For using buffered resultsets
MYSQLI_USE_RESULT	For using unbuffered resultsets
MYSQLI_ASSOC	Columns are returned into the array having the fieldname as the array index.
MYSQLI_NUM	Columns are returned into the array having an enumerated index.
MYSQLI_BOTH	Columns are returned into the array having both a numerical index and the fieldname as the associative index.
MYSQLI_NOT_NULL_FLAG	Indicates that a field is defined as NOT NULL
MYSQLI_PRI_KEY_FLAG	Field is part of a primary index
MYSQLI_UNIQUE_KEY_FLAG	Field is part of a unique index.
MYSQLI_MULTIPLE_KEY_FLAG	Field is part of an index.
MYSQLI_BLOB_FLAG	Field is defined as BLOB
MYSQLI_UNSIGNED_FLAG	Field is defined as UNSIGNED
MYSQLI_ZEROFILL_FLAG	Field is defined as ZEROFILL
MYSQLI_AUTO_INCREMENT_FLAG	Field is defined as AUTO_INCREMENT
MYSQLI_TIMESTAMP_FLAG	Field is defined as TIMESTAMP
MYSQLI_SET_FLAG	Field is defined as SET
MYSQLI_NUM_FLAG	Field is defined as NUMERIC

MYSQLI_PART_KEY_FLAG	Field is part of an multi-index
MYSQLI_GROUP_FLAG	Field is part of GROUP BY
MYSQLI_TYPE_DECIMAL	Field is defined as DECIMAL
MYSQLI_TYPE_NEWDECIMAL	Precision math DECIMAL or NUMERIC field (MySQL 5.0.3 and up)
MYSQLI_TYPE_BIT	Field is defined as BIT (MySQL 5.0.3 and up)
MYSQLI_TYPE_TINY	Field is defined as TINYINT
MYSQLI_TYPE_SHORT	Field is defined as SMALLINT
MYSQLI_TYPE_LONG	Field is defined as INT
MYSQLI_TYPE_FLOAT	Field is defined as FLOAT
MYSQLI_TYPE_DOUBLE	Field is defined as DOUBLE
MYSQLI_TYPE_NULL	Field is defined as DEFAULT NULL
MYSQLI_TYPE_TIMESTAMP	Field is defined as TIMESTAMP
MYSQLI_TYPE_LONGLONG	Field is defined as BIGINT
MYSQLI_TYPE_INT24	Field is defined as MEDIUMINT
MYSQLI_TYPE_DATE	Field is defined as DATE
MYSQLI_TYPE_TIME	Field is defined as TIME
MYSQLI_TYPE_DATETIME	Field is defined as DATETIME
MYSQLI_TYPE_YEAR	Field is defined as YEAR
MYSQLI_TYPE_NEWDATE	Field is defined as DATE
MYSQLI_TYPE_INTERVAL	Field is defined as INTERVAL
MYSQLI_TYPE_ENUM	Field is defined as ENUM
MYSQLI_TYPE_SET	Field is defined as SET
MYSQLI_TYPE_TINY_BLOB	Field is defined as TINYBLOB
MYSQLI_TYPE_MEDIUM_BLOB	Field is defined as MEDIUMBLOB
MYSQLI_TYPE_LONG_BLOB	Field is defined as LONGBLOB
MYSQLI_TYPE_BLOB	Field is defined as BLOB
MYSQLI_TYPE_VAR_STRING	Field is defined as VARCHAR
MYSQLI_TYPE_STRING	Field is defined as STRING
MYSQLI_TYPE_CHAR	Field is defined as CHAR
MYSQLI_TYPE_GEOMETRY	Field is defined as GEOMETRY
MYSQLI_NEED_DATA	More data available for bind variable
MYSQLI_NO_DATA	No more data available for bind variable
MYSQLI_DATA_TRUNCATED	Data truncation occurred. Available since PHP 5.1.0 and MySQL 5.0.5.
MYSQLI_ENUM_FLAG	Field is defined as ENUM. Available since PHP 5.3.0.

MYSQLI_CURSOR_TYPE_FOR_UPDATE
MYSQLI_CURSOR_TYPE_NO_CURSOR
MYSQLI_CURSOR_TYPE_READ_ONLY
MYSQLI_CURSOR_TYPE_SCROLLABLE
MYSQLI_STMT_ATTR_CURSOR_TYPE
MYSQLI_STMT_ATTR_PREFETCH_ROWS
MYSQLI_STMT_ATTR_UPDATE_MAX_LENGTH
MYSQLI_SET_CHARSET_NAME

2.6. The MySQLi Extension Function Summary

MySQLi Class			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
Properties			
$mysqli->affected_rows	`mysqli_affected_rows`	N/A	Gets the number of affected rows in a previous MySQL operation
$mysqli->client_info	`mysqli_get_client_info`	N/A	Returns the MySQL client version as a string
$mysqli->client_version	`mysqli_get_client_version`	N/A	Returns MySQL client version info as an integer
$mysqli->connect_errno	`mysqli_connect_errno`	N/A	Returns the error code from last connect call
$mysqli->connect_error	`mysqli_connect_error`	N/A	Returns a string description of the last connect error
$mysqli->errno	`mysqli_errno`	N/A	Returns the error code for the most recent function call
$mysqli->error	`mysqli_error`	N/A	Returns a string description of the last error
$mysqli->field_count	`mysqli_field_count`	N/A	Returns the number of columns for the most recent query
$mysqli->host_info	`mysqli_get_host_info`	N/A	Returns a string representing the type of connection used
$mysqli->protocol_version	`mysqli_get_proto_info`	N/A	Returns the version of the MySQL protocol used
$mysqli->server_info	`mysqli_get_server_info`	N/A	Returns the version of the MySQL server
$mysqli->server_version	`mysqli_get_server_version`	N/A	Returns the version of the MySQL server as an integer
$mysqli->info	`mysqli_info`	N/A	Retrieves information about the most recently executed query
$mysqli->insert_id	`mysqli_insert_id`	N/A	Returns the auto generated id used in the last query
$mysqli->sqlstate	`mysqli_sqlstate`	N/A	Returns the SQLSTATE error from previous MySQL operation
$mysqli->warning_count	`mysqli_warning_count`	N/A	Returns the number of warnings from the last query for the given link
Methods			
`mysqli->autocommit`	`mysqli_autocommit`	N/A	Turns on or off auto-commiting database modifications
`mysqli->change_user`	`mysqli_change_user`	N/A	Changes the user of the specified database connection

MySQLi Class			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
mysqli->character_set_name, mysqli->client_encoding	mysqli_character_set_name	mysqli_client_encoding	Returns the default character set for the database connection
mysqli->close	mysqli_close	N/A	Closes a previously opened database connection
mysqli->commit	mysqli_commit	N/A	Commits the current transaction
mysqli::__construct	mysqli_connect	N/A	Open a new connection to the MySQL server [Note: static (i.e. class) method]
mysqli->debug	mysqli_debug	N/A	Performs debugging operations
mysqli->dump_debug_info	mysqli_dump_debug_info	N/A	Dump debugging information into the log
mysqli->get_charset	mysqli_get_charset	N/A	Returns a character set object
mysqli->get_connection_stats	mysqli_get_connection_stats	N/A	Returns client connection statistics. Available only with mysqlnd.
mysqli->get_client_info	mysqli_get_client_info	N/A	Returns the MySQL client version as a string
mysqli->get_client_stats	mysqli_get_client_stats	N/A	Returns client per-process statistics. Available only with mysqlnd.
mysqli->get_cache_stats	mysqli_get_cache_stats	N/A	Returns client Zval cache statistics. Available only with mysqlnd.
mysqli->get_server_info	mysqli_get_server_info	N/A	NOT DOCUMENTED
mysqli->get_warnings	mysqli_get_warnings	N/A	NOT DOCUMENTED
mysqli::init	mysqli_init	N/A	Initializes MySQLi and returns a resource for use with mysqli_real_connect. [Not called on an object, as it returns a $mysqli object.]
mysqli->kill	mysqli_kill	N/A	Asks the server to kill a MySQL thread
mysqli->more_results	mysqli_more_results	N/A	Check if there are any more query results from a multi query
mysqli->multi_query	mysqli_multi_query	N/A	Performs a query on the database
mysqli->next_result	mysqli_next_result	N/A	Prepare next result from multi_query
mysqli->options	mysqli_options	mysqli_set_opt	Set options
mysqli->ping	mysqli_ping	N/A	Pings a server connection, or tries to reconnect if the connection has gone down
mysqli->prepare	mysqli_prepare	N/A	Prepare an SQL statement for execution
mysqli->query	mysqli_query	N/A	Performs a query on the database
mysqli->real_connect	mysqli_real_connect	N/A	Opens a connection to a mysql server
mysqli-	mysqli_real_escape_st	mysqli_escape_string	Escapes special characters in a

MySQLi Class			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
>real_escape_string, mysqli->escape_string	ring		string for use in an SQL statement, taking into account the current charset of the connection
mysqli->real_query	mysqli_real_query	N/A	Execute an SQL query
mysqli->rollback	mysqli_rollback	N/A	Rolls back current transaction
mysqli->select_db	mysqli_select_db	N/A	Selects the default database for database queries
mysqli->set_charset	mysqli_set_charset	N/A	Sets the default client character set
mysqli->set_local_infile_default	mysqli_set_local_infile_default	N/A	Unsets user defined handler for load local infile command
mysqli->set_local_infile_handler	mysqli_set_local_infile_handler	N/A	Set callback function for LOAD DATA LOCAL INFILE command
mysqli->ssl_set	mysqli_ssl_set	N/A	Used for establishing secure connections using SSL
mysqli->stat	mysqli_stat	N/A	Gets the current system status
mysqli->stmt_init	mysqli_stmt_init	N/A	Initializes a statement and returns an object for use with mysqli_stmt_prepare
mysqli->store_result	mysqli_store_result	N/A	Transfers a result set from the last query
mysqli->thread_id	mysqli_thread_id	N/A	Returns the thread ID for the current connection
mysqli->thread_safe	mysqli_thread_safe	N/A	Returns whether thread safety is given or not
mysqli->use_result	mysqli_use_result	N/A	Initiate a result set retrieval

MySQL_STMT			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
Properties			
$mysqli_stmt->affected_rows	mysqli_stmt_affected_rows	N/A	Returns the total number of rows changed, deleted, or inserted by the last executed statement
$mysqli_stmt->errno	mysqli_stmt_errno	N/A	Returns the error code for the most recent statement call
$mysqli_stmt->error	mysqli_stmt_error	N/A	Returns a string description for last statement error
$mysqli_stmt->field_count	mysqli_stmt_field_count	N/A	Returns the number of field in the given statement - not documented
$mysqli_stmt->insert_id	mysqli_stmt_insert_id	N/A	Get the ID generated from the previous INSERT operation
$mysqli_stmt->num_rows	mysqli_stmt_num_rows	N/A	Return the number of rows in statements result set
$mysqli_stmt->param_count	mysqli_stmt_param_count	mysqli_param_count	Returns the number of parameter for the given statement
$mysqli_stmt->sqlstate	mysqli_stmt_sqlstate	N/A	Returns SQLSTATE error from previous statement operation

MySQL_STMT			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
Methods			
mysqli_stmt->attr_get	mysqli_stmt_attr_get	N/A	Used to get the current value of a statement attribute
mysqli_stmt->attr_set	mysqli_stmt_attr_set	N/A	Used to modify the behavior of a prepared statement
mysqli_stmt->bind_param	mysqli_stmt_bind_param	mysqli_bind_param	Binds variables to a prepared statement as parameters
mysqli_stmt->bind_result	mysqli_stmt_bind_result	mysqli_bind_result	Binds variables to a prepared statement for result storage
mysqli_stmt->close	mysqli_stmt_close	N/A	Closes a prepared statement
mysqli_stmt->data_seek	mysqli_stmt_data_seek	N/A	Seeks to an arbitrary row in statement result set
mysqli_stmt->execute	mysqli_stmt_execute	mysqli_execute	Executes a prepared Query
mysqli_stmt->fetch	mysqli_stmt_fetch	mysqli_fetch	Fetch results from a prepared statement into the bound variables
mysqli_stmt->free_result	mysqli_stmt_free_result	N/A	Frees stored result memory for the given statement handle
$mysqli_stmt->get_result()	mysqli_stmt_get_result	N/A	NOT DOCUMENTED Available only with mysqlnd.
mysqli_stmt->get_warnings	mysqli_stmt_get_warnings	N/A	NOT DOCUMENTED
$mysqli_stmt->more_results()	mysqli_stmt_more_results()	N/A	NOT DOCUMENTED Available only with mysqlnd.
$mysqli_stmt->next_result()	mysqli_stmt_next_result()	N/A	NOT DOCUMENTED Available only with mysqlnd.
mysqli_stmt->num_rows	mysqli_stmt_num_rows	N/A	See also property $mysqli_stmt->num_rows
mysqli_stmt->prepare	mysqli_stmt_prepare	N/A	Prepare an SQL statement for execution
mysqli_stmt->reset	mysqli_stmt_reset	N/A	Resets a prepared statement
mysqli_stmt->result_metadata	mysqli_stmt_result_metadata	mysqli_get_metadata	Returns result set metadata from a prepared statement
mysqli_stmt->send_long_data	mysqli_stmt_send_long_data	mysqli_send_long_data	Send data in blocks
mysqli_stmt->store_result	mysqli_stmt_store_result	N/A	Transfers a result set from a prepared statement

MySQLi_RESULT			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
Properties			
$mysqli_result->current_field	mysqli_field_tell	N/A	Get current field offset of a result pointer
$mysqli_result->field_count	mysqli_num_fields	N/A	Get the number of fields in a result
$mysqli_result->lengths	mysqli_fetch_lengths	N/A	Returns the lengths of the columns of the current row in the result set
$mysqli_result->num_rows	mysqli_num_rows	N/A	Gets the number of rows in a

MySQLi_RESULT			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
			result
Methods			
mysqli_result->data_seek	mysqli_data_seek	N/A	Adjusts the result pointer to an arbitary row in the result
mysqli_result->fetch_all	mysqli_fetch_all	N/A	Fetches all result rows and returns the result set as an associative array, a numeric array, or both. Available only with mysqlnd.
mysqli_result->fetch_array	mysqli_fetch_array	N/A	Fetch a result row as an associative, a numeric array, or both
mysqli_result->fetch_assoc	mysqli_fetch_assoc	N/A	Fetch a result row as an associative array
mysqli_result->fetch_field_direct	mysqli_fetch_field_direct	N/A	Fetch meta-data for a single field
mysqli_result->fetch_field	mysqli_fetch_field	N/A	Returns the next field in the result set
mysqli_result->fetch_fields	mysqli_fetch_fields	N/A	Returns an array of objects representing the fields in a result set
mysqli_result->fetch_object	mysqli_fetch_object	N/A	Returns the current row of a result set as an object
mysqli_result->fetch_row	mysqli_fetch_row	N/A	Get a result row as an enumerated array
mysqli_result->field_seek	mysqli_field_seek	N/A	Set result pointer to a specified field offset
mysqli_result->free, mysqli_result->close, mysqli_result->free_result	mysqli_free_result	N/A	Frees the memory associated with a result

MySQL_Driver			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
Properties			
N/A			
Methods			
mysqli_driver->embedded_server_end	mysqli_embedded_server_end	N/A	NOT DOCUMENTED
mysqli_driver->embedded_server_start	mysqli_embedded_server_start	N/A	NOT DOCUMENTED

Note

Alias functions are provided for backward compatibility purposes only. Do not use them in new projects.

2.7. The MySQLi class (MySQLi)

Represents a connection between PHP and a MySQL database.

```
MySQLi {
MySQLi
```

```
    Properties
  int affected_rows ;

  string client_info ;

  int client_version ;

  string connect_errno ;

  string connect_error ;

  int errno ;

  string error ;

  int field_count ;

  int client_version ;

  string host_info ;

  string protocol_version ;

  string server_info ;

  int server_version ;

  string info ;

  mixed insert_id ;

  string sqlstate ;

  int thread_id ;

  int warning_count ;
Methods
  int mysqli_affected_rows(mysqli link);

  bool mysqli::autocommit(bool mode);

  bool mysqli::change_user(string user,
                           string password,
                           string database);

  string mysqli::character_set_name();

  string mysqli_get_client_info(mysqli link);

  int mysqli_get_client_version(mysqli link);

  bool mysqli::close();

  bool mysqli::commit();
```

```
int mysqli_connect_errno();


string mysqli_connect_error();


mysqli mysqli_connect(string host= =ini_get("mysqli.default_host"),
                      string username= =ini_get("mysqli.default_user"),
                      string passwd= =ini_get("mysqli.default_pw"),
                      string dbname= ="",
                      int port= =ini_get("mysqli.default_port"),
                      string socket= =ini_get("mysqli.default_socket"));


bool mysqli::debug(string message);


bool mysqli::dump_debug_info();


int mysqli_errno(mysqli link);


string mysqli_error(mysqli link);


int mysqli_field_count(mysqli link);


object mysqli::get_charset();


string mysqli::get_client_info();


int mysqli_get_client_version(mysqli link);


bool mysqli::get_connection_stats();


string mysqli_get_host_info(mysqli link);


int mysqli_get_proto_info(mysqli link);


string mysqli_get_server_info(mysqli link);


int mysqli_get_server_version(mysqli link);


mysqli_warning mysqli::get_warnings();


string mysqli_info(mysqli link);


mysqli mysqli::init();


mixed mysqli_insert_id(mysqli link);


bool mysqli::kill(int processid);


bool mysqli::more_results();


bool mysqli::multi_query(string query);


bool mysqli::next_result();


bool mysqli::options(int option,
                     mixed value);


bool mysqli::ping();
```

```
  public int mysqli::poll(array read,
                          array error,
                          array reject,
                          int sec,
                          int usec);


  mysqli_stmt mysqli::prepare(string query);


  mixed mysqli::query(string query,
                      int resultmode);


  bool mysqli::real_connect(string host,
                            string username,
                            string passwd,
                            string dbname,
                            int port,
                            string socket,
                            int flags);


  string mysqli::escape_string(string escapestr);


  bool mysqli::real_query(string query);


  public mysqli_result mysqli::reap_async_query();


  bool mysqli::rollback();


  bool mysqli::select_db(string dbname);


  bool mysqli::set_charset(string charset);


  void mysqli_set_local_infile_default(mysqli link);


  bool mysqli::set_local_infile_handler(mysqli link,
                                        callback read_func);


  string mysqli_sqlstate(mysqli link);


  bool mysqli::ssl_set(string key,
                       string cert,
                       string ca,
                       string capath,
                       string cipher);


  string mysqli::stat();


  mysqli_stmt mysqli::stmt_init();


  mysqli_result mysqli::store_result();


  int mysqli_thread_id(mysqli link);


  bool mysqli_thread_safe();


  mysqli_result mysqli::use_result();


  int mysqli_warning_count(mysqli link);

}
```

2.7.1. mysqli->affected_rows, mysqli_affected_rows

- mysqli->affected_rows

  mysqli_affected_rows

  Gets the number of affected rows in a previous MySQL operation

Description

Object oriented style

```
 mysqli {

  int affected_rows ;

}
```

Procedural style

```
  int mysqli_affected_rows(mysqli link);
```

Returns the number of rows affected by the last `INSERT`, `UPDATE`, `REPLACE` or `DELETE` query.

For SELECT statements `mysqli_affected_rows` works like `mysqli_num_rows`.

Parameters

link	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records where updated for an UPDATE statement, no rows matched the `WHERE` clause in the query or that no query has yet been executed. -1 indicates that the query returned an error.

> **Note**
>
> If the number of affected rows is greater than maximal int value, the number of affected rows will be returned as a string.

Examples

Example 2.1. mysqli->affected_rows example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* Insert rows */
$mysqli->query("CREATE TABLE Language SELECT * from CountryLanguage");
printf("Affected rows (INSERT): %d\n", $mysqli->affected_rows);
$mysqli->query("ALTER TABLE Language ADD Status int default 0");
/* update rows */
$mysqli->query("UPDATE Language SET Status=1 WHERE Percentage > 50");
printf("Affected rows (UPDATE): %d\n", $mysqli->affected_rows);
/* delete rows */
$mysqli->query("DELETE FROM Language WHERE Percentage < 50");
```

```
printf("Affected rows (DELETE): %d\n", $mysqli->affected_rows);
/* select all rows */
$result = $mysqli->query("SELECT CountryCode FROM Language");
printf("Affected rows (SELECT): %d\n", $mysqli->affected_rows);
$result->close();
/* Delete table Language */
$mysqli->query("DROP TABLE Language");
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
if (!$link) {
    printf("Can't connect to localhost. Error: %s\n", mysqli_connect_error());
    exit();
}
/* Insert rows */
mysqli_query($link, "CREATE TABLE Language SELECT * from CountryLanguage");
printf("Affected rows (INSERT): %d\n", mysqli_affected_rows($link));
mysqli_query($link, "ALTER TABLE Language ADD Status int default 0");
/* update rows */
mysqli_query($link, "UPDATE Language SET Status=1 WHERE Percentage > 50");
printf("Affected rows (UPDATE): %d\n", mysqli_affected_rows($link));
/* delete rows */
mysqli_query($link, "DELETE FROM Language WHERE Percentage < 50");
printf("Affected rows (DELETE): %d\n", mysqli_affected_rows($link));
/* select all rows */
$result = mysqli_query($link, "SELECT CountryCode FROM Language");
printf("Affected rows (SELECT): %d\n", mysqli_affected_rows($link));
mysqli_free_result($result);
/* Delete table Language */
mysqli_query($link, "DROP TABLE Language");
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Affected rows (INSERT): 984
Affected rows (UPDATE): 168
Affected rows (DELETE): 815
Affected rows (SELECT): 169
```

See Also

mysqli_num_rows
mysqli_info

2.7.2. mysqli::autocommit, mysqli_autocommit

- mysqli::autocommit

  mysqli_autocommit

  Turns on or off auto-commiting database modifications

Description

Object oriented style

```
bool mysqli::autocommit(bool mode);
```

Procedural style

```
bool mysqli_autocommit(mysqli link,
                       bool mode);
```

Turns on or off auto-commit mode on queries for the database connection.

To determine the current state of autocommit use the SQL command `SELECT @@autocommit`.

Parameters

link	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`
mode	Whether to turn on auto-commit or not.

Return Values

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

Notes

Note

This function doesn't work with non transactional table types (like MyISAM or ISAM).

Examples

Example 2.2. `mysqli::autocommit` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* turn autocommit on */
$mysqli->autocommit(TRUE);
if ($result = $mysqli->query("SELECT @@autocommit")) {
    $row = $result->fetch_row();
    printf("Autocommit is %s\n", $row[0]);
    $result->free();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
if (!$link) {
    printf("Can't connect to localhost. Error: %s\n", mysqli_connect_error());
    exit();
}
/* turn autocommit on */
mysqli_autocommit($link, TRUE);
if ($result = mysqli_query($link, "SELECT @@autocommit")) {
    $row = mysqli_fetch_row($result);
    printf("Autocommit is %s\n", $row[0]);
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Autocommit is 1
```

See Also

mysqli_commit
mysqli_rollback

2.7.3. mysqli::change_user, mysqli_change_user

- mysqli::change_user

  mysqli_change_user

  Changes the user of the specified database connection

Description

Object oriented style

```
  bool mysqli::change_user(string user,
                           string password,
                           string database);
```

Procedural style

```
  bool mysqli_change_user(mysqli link,
                          string user,
                          string password,
                          string database);
```

Changes the user of the specified database connection and sets the current database.

In order to successfully change users a valid *username* and *password* parameters must be provided and that user must have sufficient permissions to access the desired database. If for any reason authorization fails, the current user authentication will remain.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
user	The MySQL user name.
password	The MySQL password.
database	The database to change to. If desired, the NULL value may be passed resulting in only changing the user and not selecting a

database. To select a database in this case use the `mysqli_select_db` function.

Return Values

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

Notes

> **Note**
>
> Using this command will always cause the current database connection to behave as if was a completely new database connection, regardless of if the operation was completed successfully. This reset includes performing a rollback on any active transactions, closing all temporary tables, and unlocking all locked tables.

Examples

Example 2.3. `mysqli::change_user` example

Object oriented style

```
<?php
/* connect database test */
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* Set Variable a */
$mysqli->query("SET @a:=1");
/* reset all and select a new database */
$mysqli->change_user("my_user", "my_password", "world");
if ($result = $mysqli->query("SELECT DATABASE()")) {
    $row = $result->fetch_row();
    printf("Default database: %s\n", $row[0]);
    $result->close();
}
if ($result = $mysqli->query("SELECT @a")) {
    $row = $result->fetch_row();
    if ($row[0] === NULL) {
        printf("Value of variable a is NULL\n");
    }
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
/* connect database test */
$link = mysqli_connect("localhost", "my_user", "my_password", "test");
/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* Set Variable a */
mysqli_query($link, "SET @a:=1");
/* reset all and select a new database */
mysqli_change_user($link, "my_user", "my_password", "world");
if ($result = mysqli_query($link, "SELECT DATABASE()")) {
    $row = mysqli_fetch_row($result);
    printf("Default database: %s\n", $row[0]);
    mysqli_free_result($result);
}
if ($result = mysqli_query($link, "SELECT @a")) {
    $row = mysqli_fetch_row($result);
    if ($row[0] === NULL) {
        printf("Value of variable a is NULL\n");
    }
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Default database: world
Value of variable a is NULL
```

See Also

mysqli_connect
mysqli_select_db

2.7.4. mysqli::character_set_name, mysqli_character_set_name

- mysqli::character_set_name

  mysqli_character_set_name

  Returns the default character set for the database connection

Description

Object oriented style

```
string mysqli::character_set_name();
```

Procedural style

```
string mysqli_character_set_name(mysqli link);
```

Returns the current character set for the database connection.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

The default character set for the current connection

Examples

Example 2.4. mysqli::character_set_name example

Object oriented style

```
<?php
/* Open a connection */
```

```
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* Print current character set */
$charset = $mysqli->character_set_name();
printf ("Current character set is %s\n", $charset);
$mysqli->close();
?>
```

Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* Print current character set */
$charset = mysqli_character_set_name($link);
printf ("Current character set is %s\n",$charset);
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Current character set is latin1_swedish_ci
```

See Also

mysqli_client_encoding
mysqli_real_escape_string

2.7.5. mysqli->client_info, mysqli_get_client_info

- mysqli->client_info

  mysqli_get_client_info

  Returns the MySQL client version as a string

Description

Object oriented style

```
 mysqli {
  string client_info ;
}
```

Procedural style

```
  string mysqli_get_client_info(mysqli link);
```

Returns a string that represents the MySQL client library version.

Return Values

A string that represents the MySQL client library version

Examples

Example 2.5. mysqli_get_client_info

```
<?php
/* We don't need a connection to determine
   the version of mysql client library */
printf("Client library version: %s\n", mysqli_get_client_info());
?>
```

See Also

`mysqli_get_client_version`
`mysqli_get_server_info`
`mysqli_get_server_version`

2.7.6. `mysqli->client_version`, `mysqli_get_client_version`

- `mysqli->client_version`

  `mysqli_get_client_version`

  Get MySQL client info

Description

Object oriented style

```
 mysqli {

  int client_version ;

}
```

Procedural style

```
  int mysqli_get_client_version(mysqli link);
```

Returns client version number as an integer.

Return Values

A number that represents the MySQL client library version in format: `main_version*10000 + minor_version *100 + sub_version`. For example, 4.1.0 is returned as 40100.

This is useful to quickly determine the version of the client library to know if some capability exits.

Examples

Example 2.6. mysqli_get_client_version

```
<?php
/* We don't need a connection to determine
   the version of mysql client library */
printf("Client library version: %d\n", mysqli_get_client_version());
?>
```

See Also

mysqli_get_client_info
mysqli_get_server_info
mysqli_get_server_version

2.7.7. mysqli::close, mysqli_close

- mysqli::close

  mysqli_close

  Closes a previously opened database connection

Description

Object oriented style

```
bool mysqli::close();
```

Procedural style

```
bool mysqli_close(mysqli link);
```

Closes a previously opened database connection.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

Returns TRUE on success or FALSE on failure.

Examples

See mysqli_connect.

See Also

mysqli_connect
mysqli_init
mysqli_real_connect

2.7.8. mysqli::commit, mysqli_commit

- mysqli::commit

  mysqli_commit

  Commits the current transaction

Description

Object oriented style

```
bool mysqli::commit();
```

Procedural style

```
bool mysqli_commit(mysqli link);
```

Commits the current transaction for the database connection.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

Returns TRUE on success or FALSE on failure.

Examples

Example 2.7. mysqli::commit example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TABLE Language LIKE CountryLanguage");
/* set autocommit to off */
$mysqli->autocommit(FALSE);
/* Insert some values */
$mysqli->query("INSERT INTO Language VALUES ('DEU', 'Bavarian', 'F', 11.2)");
$mysqli->query("INSERT INTO Language VALUES ('DEU', 'Swabian', 'F', 9.4)");
/* commit transaction */
$mysqli->commit();
/* drop table */
$mysqli->query("DROP TABLE Language");
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "test");
/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* set autocommit to off */
mysqli_autocommit($link, FALSE);
mysqli_query($link, "CREATE TABLE Language LIKE CountryLanguage");
/* Insert some values */
mysqli_query($link, "INSERT INTO Language VALUES ('DEU', 'Bavarian', 'F', 11.2)");
mysqli_query($link, "INSERT INTO Language VALUES ('DEU', 'Swabian', 'F', 9.4)");
/* commit transaction */
mysqli_commit($link);
/* close connection */
mysqli_close($link);
?>
```

See Also

mysqli_autocommit
mysqli_rollback

2.7.9. mysqli->connect_errno, mysqli_connect_errno

- mysqli->connect_errno

  mysqli_connect_errno

  Returns the error code from last connect call

Description

Object oriented style

```
 mysqli {

  string connect_errno ;

}
```

Procedural style

```
  int mysqli_connect_errno();
```

Returns the last error code number from the last call to `mysqli_connect`.

> **Note**
>
> Client error message numbers are listed in the MySQL `errmsg.h` header file, server error message numbers are listed in `mysqld_error.h`. In the MySQL source distribution you can find a complete list of error messages and error numbers in the file `Docs/mysqld_error.txt`.

Return Values

An error code value for the last call to `mysqli_connect`, if it failed. zero means no error occurred.

Examples

Example 2.8. mysqli->connect_errno example

Object oriented style

```
<?php
$mysqli = @new mysqli('localhost', 'fake_user', 'my_password', 'my_db');
if ($mysqli->connect_errno) {
    die('Connect Error: ' . $mysqli->connect_errno);
}
?>
```

Procedural style

```
<?php
$link = @mysqli_connect('localhost', 'fake_user', 'my_password', 'my_db');
if (!$link) {
    die('Connect Error: ' . mysqli_connect_errno());
}
?>
```

The above examples will output:

```
Connect Error: 1045
```

See Also

mysqli_connect
mysqli_connect_error
mysqli_errno
mysqli_error
mysqli_sqlstate

2.7.10. mysqli->connect_error, mysqli_connect_error

- mysqli->connect_error

  mysqli_connect_error

  Returns a string description of the last connect error

Description

Object oriented style

```
 mysqli {

  string connect_error ;

}
```

Procedural style

```
  string mysqli_connect_error();
```

Returns the last error message string from the last call to mysqli_connect.

Return Values

A string that describes the error. NULL is returned if no error occurred.

Examples

Example 2.9. mysqli->connect_error example

Object oriented style

```
<?php
$mysqli = @new mysqli('localhost', 'fake_user', 'my_password', 'my_db');
// Works as of PHP 5.2.9 and 5.3.0.
if ($mysqli->connect_error) {
    die('Connect Error: ' . $mysqli->connect_error);
}
?>
```

Procedural style

```
<?php
$link = @mysqli_connect('localhost', 'fake_user', 'my_password', 'my_db');
if (!$link) {
    die('Connect Error: ' . mysqli_connect_error());
}
?>
```

The above examples will output:

```
Connect Error: Access denied for user 'fake_user'@'localhost' (using password: YES)
```

Notes

> **Warning**
>
> The mysqli->connect_error property only works properly as of PHP versions 5.2.9 and 5.3.0. Use the mysqli_connect_error function if compatibility with earlier PHP versions is required.

See Also

mysqli_connect
mysqli_connect_errno
mysqli_errno
mysqli_error
mysqli_sqlstate

2.7.11. mysqli::__construct, mysqli_connect

- mysqli::__construct

  mysqli_connect

  Open a new connection to the MySQL server

Description

Object oriented style

```
mysqli::__construct(string host= =ini_get("mysqli.default_host"),
                    string username= =ini_get("mysqli.default_user"),
                    string passwd= =ini_get("mysqli.default_pw"),
                    string dbname= ="",
                    int port= =ini_get("mysqli.default_port"),
                    string socket= =ini_get("mysqli.default_socket"));
```

Procedural style

```
mysqli mysqli_connect(string host= =ini_get("mysqli.default_host"),
                      string username= =ini_get("mysqli.default_user"),
                      string passwd= =ini_get("mysqli.default_pw"),
                      string dbname= ="",
                      int port= =ini_get("mysqli.default_port"),
                      string socket= =ini_get("mysqli.default_socket"));
```

Opens a connection to the MySQL Server running on.

Parameters

`host`	Can be either a host name or an IP address. Passing the `NULL` value or the string "localhost" to this parameter, the local host is assumed. When possible, pipes will be used instead of the TCP/IP protocol. Prepending host by `p:` opens a persistent connection. `mysqli_change_user` is automatically called on connections opened from the connection pool.
`username`	The MySQL user name.
`passwd`	If not provided or `NULL` , the MySQL server will attempt to authenticate the user against those user records which have no password only. This allows one username to be used with different permissions (depending on if a password as provided or not).
`dbname`	If provided will specify the default database to be used when performing queries.
`port`	Specifies the port number to attempt to connect to the MySQL server.
`socket`	Specifies the socket or named pipe that should be used.

Note

Specifying the *`socket`* parameter will not explicitly determine the type of connection to be used when connecting to the MySQL server. How the connection is made to the MySQL database is determined by the *`host`* parameter.

Return Values

Returns an object which represents the connection to a MySQL Server.

Changelog

Version	Description
5.3.0	Added the ability of persistent connections.

Examples

Example 2.10. `mysqli::__construct` example

Object oriented style

```
<?php
$mysqli = new mysqli('localhost', 'my_user', 'my_password', 'my_db');
/*
 * This is the "official" OO way to do it,
 * BUT $connect_error was broken until PHP 5.2.9 and 5.3.0.
 */
if ($mysqli->connect_error) {
    die('Connect Error (' . $mysqli->connect_errno . ') '
            . $mysqli->connect_error);
}
/*
 * Use this instead of $connect_error if you need to ensure
 * compatibility with PHP versions prior to 5.2.9 and 5.3.0.
 */
if (mysqli_connect_error()) {
    die('Connect Error (' . mysqli_connect_errno() . ') '
            . mysqli_connect_error());
}
echo 'Success... ' . $mysqli->host_info . "\n";
$mysqli->close();
?>
```

Object oriented style when extending mysqli class

```
<?php
class foo_mysqli extends mysqli {
    public function __construct($host, $user, $pass, $db) {
        parent::__construct($host, $user, $pass, $db);
        if (mysqli_connect_error()) {
            die('Connect Error (' . mysqli_connect_errno() . ') '
                    . mysqli_connect_error());
        }
    }
}
$db = new foo_mysqli('localhost', 'my_user', 'my_password', 'my_db');
echo 'Success... ' . $db->host_info . "\n";
$db->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect('localhost', 'my_user', 'my_password', 'my_db');
if (!$link) {
    die('Connect Error (' . mysqli_connect_errno() . ') '
            . mysqli_connect_error());
}
echo 'Success... ' . mysqli_get_host_info($link) . "\n";
mysqli_close($link);
?>
```

The above examples will output:

```
Success... MySQL host info: localhost via TCP/IP
```

Notes

Note

OO syntax only: If a connection fails an object is still returned. To check if the connection failed then use either the `mysqli_connect_error` function or the mysqli->connect_error property as in the preceding examples.

Note

> If it is necessary to set options, such as the connection timeout, mysqli_real_connect must be used instead.

> **Note**
>
> Calling the constructor with no parameters is the same as calling mysqli_init.

> **Note**
>
> Error "Can't create TCP/IP socket (10106)" usually means that the variables_order configure directive doesn't contain character E. On Windows, if the environment is not copied the SYSTEMROOT environment variable won't be available and PHP will have problems loading Winsock.

See Also

mysqli_real_connect
mysqli_options
mysqli_connect_errno
mysqli_connect_error
mysqli_close

2.7.12. mysqli::debug, mysqli_debug

- mysqli::debug

  mysqli_debug

  Performs debugging operations

Description

Object oriented style

```
bool mysqli::debug(string message);
```

Procedural style

```
bool mysqli_debug(string message);
```

Performs debugging operations using the Fred Fish debugging library.

Parameters

message	A string representing the debugging operation to perform

Return Values

Returns TRUE .

Notes

> **Note**
>
> To use the mysqli_debug function you must compile the MySQL client library to support debugging.

Examples

Example 2.11. Generating a Trace File

```
<?php
/* Create a trace file in '/tmp/client.trace' on the local (client) machine: */
mysqli_debug("d:t:o,/tmp/client.trace");
?>
```

See Also

mysqli_dump_debug_info
mysqli_report

2.7.13. mysqli::dump_debug_info, mysqli_dump_debug_info

- mysqli::dump_debug_info

  mysqli_dump_debug_info

  Dump debugging information into the log

Description

Object oriented style

```
bool mysqli::dump_debug_info();
```

Procedural style

```
bool mysqli_dump_debug_info(mysqli link);
```

This function is designed to be executed by an user with the SUPER privilege and is used to dump debugging information into the log for the MySQL Server relating to the connection.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

Returns TRUE on success or FALSE on failure.

See Also

mysqli_debug

2.7.14. mysqli->errno, mysqli_errno

- mysqli->errno

mysqli_errno

Returns the error code for the most recent function call

Description

Object oriented style

```
 mysqli {

  int errno ;

}
```

Procedural style

```
  int mysqli_errno(mysqli link);
```

Returns the last error code for the most recent MySQLi function call that can succeed or fail.

Client error message numbers are listed in the MySQL `errmsg.h` header file, server error message numbers are listed in `mysqld_error.h`. In the MySQL source distribution you can find a complete list of error messages and error numbers in the file `Docs/mysqld_error.txt`.

Parameters

`link`	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

An error code value for the last call, if it failed. zero means no error occurred.

Examples

Example 2.12. `mysqli->errno` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if (!$mysqli->query("SET a=1")) {
    printf("Errorcode: %d\n", $mysqli->errno);
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if (!mysqli_query($link, "SET a=1")) {
    printf("Errorcode: %d\n", mysqli_errno($link));
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Errorcode: 1193
```

See Also

mysqli_connect_errno
mysqli_connect_error
mysqli_error
mysqli_sqlstate

2.7.15. mysqli->error, mysqli_error

- mysqli->error

  mysqli_error

  Returns a string description of the last error

Description

Object oriented style

```
 mysqli {

  string error ;

}
```

Procedural style

```
  string mysqli_error(mysqli link);
```

Returns the last error message for the most recent MySQLi function call that can succeed or fail.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

A string that describes the error. An empty string if no error occurred.

Examples

Example 2.13. mysqli->error example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if (!$mysqli->query("SET a=1")) {
    printf("Errormessage: %s\n", $mysqli->error);
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if (!mysqli_query($link, "SET a=1")) {
    printf("Errormessage: %s\n", mysqli_error($link));
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Errormessage: Unknown system variable 'a'
```

See Also

mysqli_connect_errno
mysqli_connect_error
mysqli_errno
mysqli_sqlstate

2.7.16. mysqli->field_count, mysqli_field_count

- mysqli->field_count

  mysqli_field_count

  Returns the number of columns for the most recent query

Description

Object oriented style

```
mysqli_result {

 int field_count ;
}
```

Procedural style

```
  int mysqli_field_count(mysqli link);
```

Returns the number of columns for the most recent query on the connection represented by the *link* parameter. This function can be useful when using the mysqli_store_result function to determine if the query should have produced a non-empty result set or not without knowing the nature of the query.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

An integer representing the number of fields in a result set.

Examples

Example 2.14. mysqli->field_count example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");
$mysqli->query( "DROP TABLE IF EXISTS friends");
$mysqli->query( "CREATE TABLE friends (id int, name varchar(20))");
$mysqli->query( "INSERT INTO friends VALUES (1,'Hartmut'), (2, 'Ulf')");
$mysqli->real_query("SELECT * FROM friends");
if ($mysqli->field_count) {
    /* this was a select/show or describe query */
    $result = $mysqli->store_result();
    /* process resultset */
    $row = $result->fetch_row();
    /* free resultset */
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "test");
mysqli_query($link, "DROP TABLE IF EXISTS friends");
mysqli_query($link, "CREATE TABLE friends (id int, name varchar(20))");
mysqli_query($link, "INSERT INTO friends VALUES (1,'Hartmut'), (2, 'Ulf')");
mysqli_real_query($link, "SELECT * FROM friends");
if (mysqli_field_count($link)) {
    /* this was a select/show or describe query */
    $result = mysqli_store_result($link);
    /* process resultset */
    $row = mysqli_fetch_row($result);
    /* free resultset */
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

2.7.17. mysqli::get_charset, mysqli_get_charset

- `mysqli::get_charset`

  `mysqli_get_charset`

  Returns a character set object

Description

Object oriented style

```
object mysqli::get_charset();
```

Procedural style

```
object mysqli_get_charset(mysqli link);
```

Returns a character set object providing several properties of the current active character set.

Parameters

link	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

The function returns a character set object with the following properties:

charset	Character set name
collation	Collation name
dir	Directory the charset description was fetched from (?) or "" for built-in character sets
min_length	Minimum character length in bytes
max_length	Maximum character length in bytes
number	Internal character set number
state	Character set status (?)

Examples

Example 2.15. `mysqli::get_charset` example

Object oriented style

```
<?php
  $db = mysqli_init();
  $db->real_connect("localhost","root","","test");
  var_dump($db->get_charset());
?>
```

Procedural style

```
<?php
  $db = mysqli_init();
  mysqli_real_connect($db, "localhost","root","","test");
  var_dump($db->get_charset());
?>
```

The above examples will output:

```
object(stdClass)#2 (7) {
  ["charset"]=>
  string(6) "latin1"
  ["collation"]=>
  string(17) "latin1_swedish_ci"
  ["dir"]=>
  string(0) ""
  ["min_length"]=>
  int(1)
  ["max_length"]=>
  int(1)
  ["number"]=>
  int(8)
  ["state"]=>
  int(801)
}
```

See Also

mysqli_character_set_name
mysqli_set_charset

2.7.18. mysqli->get_client_info, mysqli_get_client_info

- mysqli->get_client_info

  mysqli_get_client_info

  Returns the MySQL client version as a string

Description

Object oriented style

```
string mysqli::get_client_info();
```

Procedural style

```
string mysqli_get_client_info(mysqli link);
```

Returns a string that represents the MySQL client library version.

Return Values

A string that represents the MySQL client library version

Examples

Example 2.16. mysqli_get_client_info

```
<?php
/* We don't need a connection to determine
   the version of mysql client library */
printf("Client library version: %s\n", mysqli_get_client_info());
?>
```

See Also

mysqli_get_client_version
mysqli_get_server_info
mysqli_get_server_version

2.7.19. mysqli->client_version, mysqli_get_client_version

- mysqli->client_version

  mysqli_get_client_version

  Get MySQL client info

Description

Object oriented style

```
 mysqli {

  int client_version ;

}
```

Procedural style

```
  int mysqli_get_client_version(mysqli link);
```

Returns client version number as an integer.

Return Values

A number that represents the MySQL client library version in format: `main_version*10000 + minor_version *100 + sub_version`. For example, 4.1.0 is returned as 40100.

This is useful to quickly determine the version of the client library to know if some capability exits.

Examples

Example 2.17. mysqli_get_client_version

```
<?php
/* We don't need a connection to determine
   the version of mysql client library */
printf("Client library version: %d\n", mysqli_get_client_version());
?>
```

See Also

mysqli_get_client_info
mysqli_get_server_info
mysqli_get_server_version

2.7.20. mysqli::get_connection_stats, mysqli_get_connection_stats

- mysqli::get_connection_stats

  mysqli_get_connection_stats

  Returns statistics about the client connection

Description

Object oriented style

```
bool mysqli::get_connection_stats();
```

Procedural style

```
array mysqli_get_connection_stats(mysqli link);
```

Warning

This function is currently not documented; only its argument list is available.

Returns statistics about the client connection. Available only with mysqlnd.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

Returns an array with connection stats if success, FALSE otherwise.

Examples

Example 2.18. A mysqli_get_connection_stats example

```
<?php
$link = mysqli_connect();
print_r(mysqli_get_connection_stats($link));
?>
```

The above example will output something similar to:

```
Array
(
    [bytes_sent] => 43
    [bytes_received] => 80
    [packets_sent] => 1
    [packets_received] => 2
    [protocol_overhead_in] => 8
    [protocol_overhead_out] => 4
    [bytes_received_ok_packet] => 11
    [bytes_received_eof_packet] => 0
    [bytes_received_rset_header_packet] => 0
    [bytes_received_rset_field_meta_packet] => 0
    [bytes_received_rset_row_packet] => 0
    [bytes_received_prepare_response_packet] => 0
    [bytes_received_change_user_packet] => 0
    [packets_sent_command] => 0
    [packets_received_ok] => 1
    [packets_received_eof] => 0
    [packets_received_rset_header] => 0
    [packets_received_rset_field_meta] => 0
    [packets_received_rset_row] => 0
    [packets_received_prepare_response] => 0
    [packets_received_change_user] => 0
    [result_set_queries] => 0
    [non_result_set_queries] => 0
    [no_index_used] => 0
    [bad_index_used] => 0
    [slow_queries] => 0
    [buffered_sets] => 0
    [unbuffered_sets] => 0
    [ps_buffered_sets] => 0
    [ps_unbuffered_sets] => 0
    [flushed_normal_sets] => 0
    [flushed_ps_sets] => 0
    [ps_prepared_never_executed] => 0
    [ps_prepared_once_executed] => 0
    [rows_fetched_from_server_normal] => 0
    [rows_fetched_from_server_ps] => 0
    [rows_buffered_from_client_normal] => 0
    [rows_buffered_from_client_ps] => 0
    [rows_fetched_from_client_normal_buffered] => 0
    [rows_fetched_from_client_normal_unbuffered] => 0
    [rows_fetched_from_client_ps_buffered] => 0
    [rows_fetched_from_client_ps_unbuffered] => 0
    [rows_fetched_from_client_ps_cursor] => 0
    [rows_skipped_normal] => 0
    [rows_skipped_ps] => 0
    [copy_on_write_saved] => 0
    [copy_on_write_performed] => 0
    [command_buffer_too_small] => 0
    [connect_success] => 1
    [connect_failure] => 0
    [connection_reused] => 0
    [reconnect] => 0
    [pconnect_success] => 0
    [active_connections] => 1
    [active_persistent_connections] => 0
    [explicit_close] => 0
    [implicit_close] => 0
    [disconnect_close] => 0
    [in_middle_of_command_close] => 0
    [explicit_free_result] => 0
    [implicit_free_result] => 0
    [explicit_stmt_close] => 0
    [implicit_stmt_close] => 0
    [mem_emalloc_count] => 0
    [mem_emalloc_ammount] => 0
    [mem_ecalloc_count] => 0
    [mem_ecalloc_ammount] => 0
    [mem_erealloc_count] => 0
    [mem_erealloc_ammount] => 0
    [mem_efree_count] => 0
    [mem_malloc_count] => 0
    [mem_malloc_ammount] => 0
    [mem_calloc_count] => 0
    [mem_calloc_ammount] => 0
    [mem_realloc_count] => 0
    [mem_realloc_ammount] => 0
    [mem_free_count] => 0
    [proto_text_fetched_null] => 0
    [proto_text_fetched_bit] => 0
    [proto_text_fetched_tinyint] => 0
    [proto_text_fetched_short] => 0
    [proto_text_fetched_int24] => 0
    [proto_text_fetched_int] => 0
    [proto_text_fetched_bigint] => 0
    [proto_text_fetched_decimal] => 0
    [proto_text_fetched_float] => 0
    [proto_text_fetched_double] => 0
    [proto_text_fetched_date] => 0
    [proto_text_fetched_year] => 0
    [proto_text_fetched_time] => 0
    [proto_text_fetched_datetime] => 0
    [proto_text_fetched_timestamp] => 0
    [proto_text_fetched_string] => 0
    [proto_text_fetched_blob] => 0
    [proto_text_fetched_enum] => 0
    [proto_text_fetched_set] => 0
    [proto_text_fetched_geometry] => 0
    [proto_text_fetched_other] => 0
    [proto_binary_fetched_null] => 0
    [proto_binary_fetched_bit] => 0
    [proto_binary_fetched_tinyint] => 0
    [proto_binary_fetched_short] => 0
    [proto_binary_fetched_int24] => 0
    [proto_binary_fetched_int] => 0
    [proto_binary_fetched_bigint] => 0
    [proto_binary_fetched_decimal] => 0
    [proto_binary_fetched_float] => 0
    [proto_binary_fetched_double] => 0
    [proto_binary_fetched_date] => 0
    [proto_binary_fetched_year] => 0
    [proto_binary_fetched_time] => 0
    [proto_binary_fetched_datetime] => 0
    [proto_binary_fetched_timestamp] => 0
    [proto_binary_fetched_string] => 0
    [proto_binary_fetched_blob] => 0
    [proto_binary_fetched_enum] => 0
    [proto_binary_fetched_set] => 0
    [proto_binary_fetched_geometry] => 0
    [proto_binary_fetched_other] => 0
)
```

See Also

Stats description

2.7.21. mysqli->host_info, mysqli_get_host_info

- `mysqli->host_info`

  `mysqli_get_host_info`

  Returns a string representing the type of connection used

Description

Object oriented style

```
 mysqli {

  string host_info ;

}
```

Procedural style

```
  string mysqli_get_host_info(mysqli link);
```

Returns a string describing the connection represented by the *`link`* parameter (including the server host name).

Parameters

`link`	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

A character string representing the server hostname and the connection type.

Examples

Example 2.19. mysqli->host_info example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* print host information */
printf("Host info: %s\n", $mysqli->host_info);
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* print host information */
printf("Host info: %s\n", mysqli_get_host_info($link));
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Host info: Localhost via UNIX socket
```

See Also

mysqli_get_proto_info

2.7.22. mysqli->protocol_version, mysqli_get_proto_info

- mysqli->protocol_version

  mysqli_get_proto_info

  Returns the version of the MySQL protocol used

Description

Object oriented style

```
 mysqli {
  string protocol_version ;
}
```

Procedural style

```
  int mysqli_get_proto_info(mysqli link);
```

Returns an integer representing the MySQL protocol version used by the connection represented by the *link* parameter.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

Returns an integer representing the protocol version.

Examples

Example 2.20. mysqli->protocol_version example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* print protocol version */
printf("Protocol version: %d\n", $mysqli->protocol_version);
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* print protocol version */
printf("Protocol version: %d\n", mysqli_get_proto_info($link));
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Protocol version: 10
```

See Also

mysqli_get_host_info

2.7.23. mysqli->server_info, mysqli_get_server_info

- mysqli->server_info

  mysqli_get_server_info

  Returns the version of the MySQL server

Description

Object oriented style

```
 mysqli {
  string server_info ;
}
```

Procedural style

```
  string mysqli_get_server_info(mysqli link);
```

Returns a string representing the version of the MySQL server that the MySQLi extension is connected to.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

A character string representing the server version.

Examples

Example 2.21. mysqli->server_info example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* print server version */
printf("Server version: %s\n", $mysqli->server_info);
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* print server version */
printf("Server version: %s\n", mysqli_get_server_info($link));
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Server version: 4.1.2-alpha-debug
```

See Also

`mysqli_get_client_info`
`mysqli_get_client_version`
`mysqli_get_server_version`

2.7.24. `mysqli->server_version`, `mysqli_get_server_version`

- `mysqli->server_version`

  `mysqli_get_server_version`

  Returns the version of the MySQL server as an integer

Description

Object oriented style

```
 mysqli {

  int server_version ;

}
```

Procedural style

```
  int mysqli_get_server_version(mysqli link);
```

The `mysqli_get_server_version` function returns the version of the server connected to (represented by the *`link`* parameter) as an integer.

Parameters

`link`	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

An integer representing the server version.

The form of this version number is `main_version * 10000 + minor_version * 100 + sub_version` (i.e. version 4.1.0 is 40100).

Examples

Example 2.22. `mysqli->server_version` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* print server version */
printf("Server version: %d\n", $mysqli->server_version);
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* print server version */
printf("Server version: %d\n", mysqli_get_server_version($link));
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Server version: 40102
```

See Also

`mysqli_get_client_info`
`mysqli_get_client_version`
`mysqli_get_server_info`

2.7.25. `mysqli::get_warnings`, `mysqli_get_warnings`

- `mysqli::get_warnings`

  `mysqli_get_warnings`

  Get result of SHOW WARNINGS

Description

Object oriented style

```
mysqli_warning mysqli::get_warnings();
```

Procedural style

```
mysqli_warning mysqli_get_warnings(mysqli link);
```

Warning

This function is currently not documented; only its argument list is available.

2.7.26. mysqli->info, mysqli_info

- mysqli->info

  mysqli_info

  Retrieves information about the most recently executed query

Description

Object oriented style

```
mysqli {

 string info ;

}
```

Procedural style

```
string mysqli_info(mysqli link);
```

The `mysqli_info` function returns a string providing information about the last query executed. The nature of this string is provided below:

Table 2.2. Possible mysqli_info return values

Query type	Example result string
INSERT INTO...SELECT...	Records: 100 Duplicates: 0 Warnings: 0
INSERT INTO...VALUES (...),(...),(...)	Records: 3 Duplicates: 0 Warnings: 0
LOAD DATA INFILE ...	Records: 1 Deleted: 0 Skipped: 0 Warnings: 0
ALTER TABLE ...	Records: 3 Duplicates: 0 Warnings: 0
UPDATE ...	Rows matched: 40 Changed: 40 Warnings: 0

Note

Queries which do not fall into one of the preceding formats are not supported. In these situations, `mysqli_info` will return an empty string.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

A character string representing additional information about the most recently executed query.

Examples

Example 2.23. mysqli->info example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TEMPORARY TABLE t1 LIKE City");
/* INSERT INTO .. SELECT */
$mysqli->query("INSERT INTO t1 SELECT * FROM City ORDER BY ID LIMIT 150");
printf("%s\n", $mysqli->info);
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TEMPORARY TABLE t1 LIKE City");
/* INSERT INTO .. SELECT */
mysqli_query($link, "INSERT INTO t1 SELECT * FROM City ORDER BY ID LIMIT 150");
printf("%s\n", mysqli_info($link));
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Records: 150  Duplicates: 0  Warnings: 0
```

See Also

mysqli_affected_rows
mysqli_warning_count
mysqli_num_rows

2.7.27. mysqli::init, mysqli_init

- mysqli::init

mysqli_init

Initializes MySQLi and returns a resource for use with mysqli_real_connect()

Description

Object oriented style

```
mysqli mysqli::init();
```

Procedural style

```
mysqli mysqli_init();
```

Allocates or initializes a MYSQL object suitable for mysqli_options and mysqli_real_connect.

> **Note**
>
> Any subsequent calls to any mysqli function (except mysqli_options) will fail until mysqli_real_connect was called.

Return Values

Returns an object.

Examples

See mysqli_real_connect.

See Also

mysqli_options
mysqli_close
mysqli_real_connect
mysqli_connect

2.7.28. mysqli->insert_id, mysqli_insert_id

- mysqli->insert_id

  mysqli_insert_id

  Returns the auto generated id used in the last query

Description

Object oriented style

```
mysqli {
  mixed insert_id ;
}
```

Procedural style

```
mixed mysqli_insert_id(mysqli link);
```

The mysqli_insert_id function returns the ID generated by a query on a table with a column having the AUTO_INCREMENT attribute. If the last query wasn't an INSERT or UPDATE statement or if the modified table does not have a column with the AUTO_INCREMENT attribute, this function will return zero.

Note

Performing an INSERT or UPDATE statement using the LAST_INSERT_ID() function will also modify the value returned by the mysqli_insert_id function.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

The value of the AUTO_INCREMENT field that was updated by the previous query. Returns zero if there was no previous query on the connection or if the query did not update an AUTO_INCREMENT value.

Note

If the number is greater than maximal int value, mysqli_insert_id will return a string.

Examples

Example 2.24. mysqli->insert_id example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TABLE myCity LIKE City");
$query = "INSERT INTO myCity VALUES (NULL, 'Stuttgart', 'DEU', 'Stuttgart', 617000)";
$mysqli->query($query);
printf ("New Record has id %d.\n", $mysqli->insert_id);
/* drop table */
$mysqli->query("DROP TABLE myCity");
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TABLE myCity LIKE City");
$query = "INSERT INTO myCity VALUES (NULL, 'Stuttgart', 'DEU', 'Stuttgart', 617000)";
mysqli_query($link, $query);
printf ("New Record has id %d.\n", mysqli_insert_id($link));
/* drop table */
mysqli_query($link, "DROP TABLE myCity");
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
New Record has id 1.
```

2.7.29. mysqli::kill, mysqli_kill

- mysqli::kill

  mysqli_kill

  Asks the server to kill a MySQL thread

Description

Object oriented style

```
bool mysqli::kill(int processid);
```

Procedural style

```
bool mysqli_kill(mysqli link,
                 int processid);
```

This function is used to ask the server to kill a MySQL thread specified by the *processid* parameter. This value must be retrieved by calling the mysqli_thread_id function.

To stop a running query you should use the SQL command KILL QUERY processid.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

Returns TRUE on success or FALSE on failure.

Examples

Example 2.25. mysqli::kill example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* determine our thread id */
$thread_id = $mysqli->thread_id;
/* Kill connection */
$mysqli->kill($thread_id);
/* This should produce an error */
if (!$mysqli->query("CREATE TABLE myCity LIKE City")) {
    printf("Error: %s\n", $mysqli->error);
    exit;
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* determine our thread id */
$thread_id = mysqli_thread_id($link);
/* Kill connection */
mysqli_kill($link, $thread_id);
/* This should produce an error */
if (!mysqli_query($link, "CREATE TABLE myCity LIKE City")) {
    printf("Error: %s\n", mysqli_error($link));
    exit;
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Error: MySQL server has gone away
```

See Also

`mysqli_thread_id`

2.7.30. `mysqli::more_results`, `mysqli_more_results`

- `mysqli::more_results`

  `mysqli_more_results`

  Check if there are any more query results from a multi query

Description

Object oriented style

```
bool mysqli::more_results();
```

Procedural style

```
bool mysqli_more_results(mysqli link);
```

Indicates if one or more result sets are available from a previous call to `mysqli_multi_query`.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

Returns TRUE on success or FALSE on failure.

Examples

See mysqli_multi_query.

See Also

mysqli_multi_query
mysqli_next_result
mysqli_store_result
mysqli_use_result

2.7.31. mysqli::multi_query, mysqli_multi_query

- mysqli::multi_query

  mysqli_multi_query

  Performs a query on the database

Description

Object oriented style

```
bool mysqli::multi_query(string query);
```

Procedural style

```
bool mysqli_multi_query(mysqli link,
                        string query);
```

Executes one or multiple queries which are concatenated by a semicolon.

To retrieve the resultset from the first query you can use mysqli_use_result or mysqli_store_result. All subsequent query results can be processed using mysqli_more_results and mysqli_next_result.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
query	The query, as a string. Data inside the query should be properly escaped.

Return Values

Returns FALSE if the first statement failed. To retrieve subsequent errors from other statements you have to call mysqli_next_result first.

Examples

Example 2.26. mysqli::multi_query example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query  = "SELECT CURRENT_USER();";
$query .= "SELECT Name FROM City ORDER BY ID LIMIT 20, 5";
/* execute multi query */
if ($mysqli->multi_query($query)) {
    do {
        /* store first result set */
        if ($result = $mysqli->store_result()) {
            while ($row = $result->fetch_row()) {
                printf("%s\n", $row[0]);
            }
            $result->free();
        }
        /* print divider */
        if ($mysqli->more_results()) {
            printf("-----------------\n");
        }
    } while ($mysqli->next_result());
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query  = "SELECT CURRENT_USER();";
$query .= "SELECT Name FROM City ORDER BY ID LIMIT 20, 5";
/* execute multi query */
if (mysqli_multi_query($link, $query)) {
    do {
        /* store first result set */
        if ($result = mysqli_store_result($link)) {
            while ($row = mysqli_fetch_row($result)) {
                printf("%s\n", $row[0]);
            }
            mysqli_free_result($result);
        }
        /* print divider */
        if (mysqli_more_results($link)) {
            printf("-----------------\n");
        }
    } while (mysqli_next_result($link));
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output something similar to:

```
my_user@localhost
-----------------
Amersfoort
Maastricht
Dordrecht
Leiden
Haarlemmermeer
```

```
```

See Also

mysqli_use_result
mysqli_store_result
mysqli_next_result
mysqli_more_results

2.7.32. mysqli::next_result, mysqli_next_result

- mysqli::next_result

  mysqli_next_result

  Prepare next result from multi_query

Description

Object oriented style

```
bool mysqli::next_result();
```

Procedural style

```
bool mysqli_next_result(mysqli link);
```

Prepares next result set from a previous call to `mysqli_multi_query` which can be retrieved by `mysqli_store_result` or `mysqli_use_result`.

Parameters

link	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

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

Examples

See `mysqli_multi_query`.

See Also

mysqli_multi_query
mysqli_more_results
mysqli_store_result
mysqli_use_result

2.7.33. mysqli::options, mysqli_options

- mysqli::options

  mysqli_options

  Set options

Description

Object oriented style

```
bool mysqli::options(int option,
                     mixed value);
```

Procedural style

```
bool mysqli_options(mysqli link,
                    int option,
                    mixed value);
```

Used to set extra connect options and affect behavior for a connection.

This function may be called multiple times to set several options.

mysqli_options should be called after mysqli_init and before mysqli_real_connect.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
option	The option that you want to set. It can be one of the following values:

Table 2.3. Valid options

Name	Description
MYSQLI_OPT_CONNECT_TIMEOUT	connection timeout in seconds (supported on Windows with TCP/IP since PHP 5.3.1)
MYSQLI_OPT_LOCAL_INFILE	enable/disable use of LOAD LOCAL INFILE
MYSQLI_INIT_COMMAND	command to execute after when connecting to MySQL server
MYSQLI_READ_DEFAULT_FILE	Read options from named option file instead of my.cnf
MYSQLI_READ_DEFAULT_GROUP	Read options from the named group from my.cnf or the file specified with MYSQL_READ_DEFAULT_FILE .

value	The value for the option.

Return Values

Returns TRUE on success or FALSE on failure.

Examples

See mysqli_real_connect.

See Also

mysqli_init
mysqli_real_connect

2.7.34. mysqli::ping, mysqli_ping

- mysqli::ping

  mysqli_ping

  Pings a server connection, or tries to reconnect if the connection has gone down

Description

Object oriented style

```
bool mysqli::ping();
```

Procedural style

```
bool mysqli_ping(mysqli link);
```

Checks whether the connection to the server is working. If it has gone down, and global option mysqli.reconnect is enabled an automatic reconnection is attempted.

This function can be used by clients that remain idle for a long while, to check whether the server has closed the connection and reconnect if necessary.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

Returns TRUE on success or FALSE on failure.

Examples

Example 2.27. mysqli::ping example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* check if server is alive */
if ($mysqli->ping()) {
    printf ("Our connection is ok!\n");
} else {
    printf ("Error: %s\n", $mysqli->error);
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
```

```
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* check if server is alive */
if (mysqli_ping($link)) {
    printf ("Our connection is ok!\n");
} else {
    printf ("Error: %s\n", mysqli_error($link));
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Our connection is ok!
```

2.7.35. mysqli::poll, mysqli_poll

- mysqli::poll

  mysqli_poll

  Poll connections

Description

Object oriented style

```
  public int mysqli::poll(array read,
                          array error,
                          array reject,
                          int sec,
                          int usec);
```

Procedural style

```
  int mysqli_poll(array read,
                  array error,
                  array reject,
                  int sec,
                  int usec);
```

Warning

This function is currently not documented; only its argument list is available.

Poll connections. Available only with mysqlnd.

Parameters

read

error

reject

sec Number of seconds to wait, must be non-negative.

usec	Number of microseconds to wait, must be non-negative.

Return Values

Returns number of ready connections in success, FALSE otherwise.

Examples

Example 2.28. A mysqli_poll example

```
<?php
$link1 = mysqli_connect();
$link1->query("SELECT 'test'", MYSQLI_ASYNC);
$all_links = array($link1);
$processed = 0;
do {
    $links = $errors = $reject = array();
    foreach ($all_links as $link) {
        $links[] = $errors[] = $reject[] = $link;
    }
    if (!mysqli_poll($links, $errors, $reject, 1)) {
        continue;
    }
    foreach ($links as $link) {
        if ($result = $link->reap_async_query()) {
            print_r($result->fetch_row());
            mysqli_free_result($result);
            $processed++;
        }
    }
} while ($processed < count($all_links));
?>
```

The above example will output:

```
Array
(
    [0] => test
)
```

See Also

2.7.36. mysqli::prepare, mysqli_prepare

- mysqli::prepare

  mysqli_prepare

  Prepare an SQL statement for execution

Description

Object oriented style

```
mysqli_stmt mysqli::prepare(string query);
```

Procedural style

```
mysqli_stmt mysqli_prepare(mysqli link,
                           string query);
```

Prepares the SQL query, and returns a statement handle to be used for further operations on the statement. The query must consist of a single SQL statement.

The parameter markers must be bound to application variables using `mysqli_stmt_bind_param` and/or `mysqli_stmt_bind_result` before executing the statement or fetching rows.

Parameters

link	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`
query	The query, as a string.

Note

You should not add a terminating semicolon or `\g` to the statement.

This parameter can include one or more parameter markers in the SQL statement by embedding question mark (`?`) characters at the appropriate positions.

Note

The markers are legal only in certain places in SQL statements. For example, they are allowed in the `VALUES()` list of an `INSERT` statement (to specify column values for a row), or in a comparison with a column in a `WHERE` clause to specify a comparison value.

However, they are not allowed for identifiers (such as table or column names), in the select list that names the columns to be returned by a `SELECT` statement, or to specify both operands of a binary operator such as the `=` equal sign. The latter restriction is necessary because it would be impossible to determine the parameter type. It's not allowed to compare marker with `NULL` by `? IS NULL` too. In general, parameters are legal only in Data Manipulation Language (DML) statements, and not in Data Definition Language (DDL) statements.

Return Values

`mysqli_prepare` returns a statement object or `FALSE` if an error occurred.

Examples

Example 2.29. `mysqli::prepare` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$city = "Amersfoort";
/* create a prepared statement */
if ($stmt = $mysqli->prepare("SELECT District FROM City WHERE Name=?")) {
    /* bind parameters for markers */
    $stmt->bind_param("s", $city);
    /* execute query */
    $stmt->execute();
    /* bind result variables */
    $stmt->bind_result($district);
    /* fetch value */
    $stmt->fetch();
    printf("%s is in district %s\n", $city, $district);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$city = "Amersfoort";
/* create a prepared statement */
if ($stmt = mysqli_prepare($link, "SELECT District FROM City WHERE Name=?")) {
    /* bind parameters for markers */
    mysqli_stmt_bind_param($stmt, "s", $city);
    /* execute query */
    mysqli_stmt_execute($stmt);
    /* bind result variables */
    mysqli_stmt_bind_result($stmt, $district);
    /* fetch value */
    mysqli_stmt_fetch($stmt);
    printf("%s is in district %s\n", $city, $district);
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Amersfoort is in district Utrecht
```

See Also

mysqli_stmt_execute
mysqli_stmt_fetch
mysqli_stmt_bind_param
mysqli_stmt_bind_result
mysqli_stmt_close

2.7.37. mysqli::query, mysqli_query

- mysqli::query

  mysqli_query

  Performs a query on the database

Description

Object oriented style

```
  mixed mysqli::query(string query,
                      int resultmode);
```

Procedural style

```
mixed mysqli_query(mysqli link,
                   string query,
                   int resultmode);
```

Performs a *query* against the database.

Functionally, using this function is identical to calling mysqli_real_query followed either by mysqli_use_result or mysqli_store_result.

Note

In the case where you pass a statement to mysqli_query that is longer than max_allowed_packet of the server, the returned error codes are different depending on whether you are using MySQL Native Driver (mysqlnd) or MySQL Client Library (libmysql). The behavior is as follows:

- mysqlnd on Linux returns an error code of 1153. The error message means “got a packet bigger than max_allowed_packet bytes”.
- mysqlnd on Windows returns an error code 2006. This error message means “server has gone away”.
- libmysql on all platforms returns an error code 2006. This error message means “server has gone away”.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
query	The query string. Data inside the query should be properly escaped.
resultmode	Either the constant MYSQLI_USE_RESULT or MYSQLI_STORE_RESULT depending on the desired behavior. By default, MYSQLI_STORE_RESULT is used. If you use MYSQLI_USE_RESULT all subsequent calls will return error Commands out of sync unless you call mysqli_free_result With MYSQLI_ASYNC (available with mysqlnd), it is possible to perform query asynchronously. mysqli_poll is then used to get results from such queries.

Return Values

Returns FALSE on failure. For successful SELECT, SHOW, DESCRIBE or EXPLAIN queries mysqli_query will return a result object. For other successful queries mysqli_query will return TRUE .

Changelog

Version	Description
5.3.0	Added the ability of async queries.

Examples

Example 2.30. mysqli::query example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* Create table doesn't return a resultset */
if ($mysqli->query("CREATE TEMPORARY TABLE myCity LIKE City") === TRUE) {
    printf("Table myCity successfully created.\n");
}
/* Select queries return a resultset */
if ($result = $mysqli->query("SELECT Name FROM City LIMIT 10")) {
    printf("Select returned %d rows.\n", $result->num_rows);
    /* free result set */
    $result->close();
}
/* If we have to retrieve large amount of data we use MYSQLI_USE_RESULT */
if ($result = $mysqli->query("SELECT * FROM City", MYSQLI_USE_RESULT)) {
    /* Note, that we can't execute any functions which interact with the
       server until result set was closed. All calls will return an
       'out of sync' error */
    if (!$mysqli->query("SET @a:='this will not work'")) {
        printf("Error: %s\n", $mysqli->error);
    }
    $result->close();
}
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* Create table doesn't return a resultset */
if (mysqli_query($link, "CREATE TEMPORARY TABLE myCity LIKE City") === TRUE) {
    printf("Table myCity successfully created.\n");
}
/* Select queries return a resultset */
if ($result = mysqli_query($link, "SELECT Name FROM City LIMIT 10")) {
    printf("Select returned %d rows.\n", mysqli_num_rows($result));
    /* free result set */
    mysqli_free_result($result);
}
/* If we have to retrieve large amount of data we use MYSQLI_USE_RESULT */
if ($result = mysqli_query($link, "SELECT * FROM City", MYSQLI_USE_RESULT)) {
    /* Note, that we can't execute any functions which interact with the
       server until result set was closed. All calls will return an
       'out of sync' error */
    if (!mysqli_query($link, "SET @a:='this will not work'")) {
        printf("Error: %s\n", mysqli_error($link));
    }
    mysqli_free_result($result);
}
mysqli_close($link);
?>
```

The above examples will output:

```
Table myCity successfully created.
Select returned 10 rows.
Error: Commands out of sync;  You can't run this command now
```

See Also

`mysqli_real_query`

mysqli_multi_query
mysqli_free_result

2.7.38. mysqli::real_connect, mysqli_real_connect

- mysqli::real_connect

  mysqli_real_connect

  Opens a connection to a mysql server

Description

Object oriented style

```
bool mysqli::real_connect(string host,
                          string username,
                          string passwd,
                          string dbname,
                          int port,
                          string socket,
                          int flags);
```

Procedural style

```
bool mysqli_real_connect(mysqli link,
                         string host,
                         string username,
                         string passwd,
                         string dbname,
                         int port,
                         string socket,
                         int flags);
```

Establish a connection to a MySQL database engine.

This function differs from mysqli_connect:

- mysqli_real_connect needs a valid object which has to be created by function mysqli_init.
- With the mysqli_options function you can set various options for connection.
- There is a *flags* parameter.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
host	Can be either a host name or an IP address. Passing the NULL value or the string "localhost" to this parameter, the local host is assumed. When possible, pipes will be used instead of the TCP/IP protocol.
username	The MySQL user name.
passwd	If provided or NULL , the MySQL server will attempt to authenticate the user against those user records which have no password only. This allows one username to be used with different permissions (depending on if a password as provided or not).
dbname	If provided will specify the default database to be used when performing queries.
port	Specifies the port number to attempt to connect to the MySQL server.

socket	Specifies the socket or named pipe that should be used.

> **Note**
>
> Specifying the *socket* parameter will not explicitly determine the type of connection to be used when connecting to the MySQL server. How the connection is made to the MySQL database is determined by the *host* parameter.

flags	With the parameter *flags* you can set different connection options:

Table 2.4. Supported flags

Name	Description
MYSQLI_CLIENT_COMPRESS	Use compression protocol
MYSQLI_CLIENT_FOUND_ROWS	return number of matched rows, not the number of affected rows
MYSQLI_CLIENT_IGNORE_SPACE	Allow spaces after function names. Makes all function names reserved words.
MYSQLI_CLIENT_INTERACTIVE	Allow interactive_timeout seconds (instead of wait_timeout seconds) of inactivity before closing the connection
MYSQLI_CLIENT_SSL	Use SSL (encryption)

> **Note**
>
> For security reasons the MULTI_STATEMENT flag is not supported in PHP. If you want to execute multiple queries use the mysqli_multi_query function.

Return Values

Returns TRUE on success or FALSE on failure.

Examples

Example 2.31. mysqli::real_connect example

Object oriented style

```
<?php
$mysqli = mysqli_init();
if (!$mysqli) {
    die('mysqli_init failed');
}
if (!$mysqli->options(MYSQLI_INIT_COMMAND, 'SET AUTOCOMMIT = 0')) {
    die('Setting MYSQLI_INIT_COMMAND failed');
}
if (!$mysqli->options(MYSQLI_OPT_CONNECT_TIMEOUT, 5)) {
    die('Setting MYSQLI_OPT_CONNECT_TIMEOUT failed');
}
if (!$mysqli->real_connect('localhost', 'my_user', 'my_password', 'my_db')) {
    die('Connect Error (' . mysqli_connect_errno() . ') '
            . mysqli_connect_error());
}
echo 'Success... ' . $mysqli->host_info . "\n";
$mysqli->close();
?>
```

Object oriented style when extending mysqli class

```
<?php
class foo_mysqli extends mysqli {
    public function __construct($host, $user, $pass, $db) {
        parent::init();
        if (!parent::options(MYSQLI_INIT_COMMAND, 'SET AUTOCOMMIT = 0')) {
            die('Setting MYSQLI_INIT_COMMAND failed');
        }
        if (!parent::options(MYSQLI_OPT_CONNECT_TIMEOUT, 5)) {
            die('Setting MYSQLI_OPT_CONNECT_TIMEOUT failed');
        }
        if (!parent::real_connect($host, $user, $pass, $db)) {
            die('Connect Error (' . mysqli_connect_errno() . ') '
                    . mysqli_connect_error());
        }
    }
}
$db = new foo_mysqli('localhost', 'my_user', 'my_password', 'my_db');
echo 'Success... ' . $db->host_info . "\n";
$db->close();
?>
```

Procedural style

```
<?php
$link = mysqli_init();
if (!$link) {
    die('mysqli_init failed');
}
if (!mysqli_options($link, MYSQLI_INIT_COMMAND, 'SET AUTOCOMMIT = 0')) {
    die('Setting MYSQLI_INIT_COMMAND failed');
}
if (!mysqli_options($link, MYSQLI_OPT_CONNECT_TIMEOUT, 5)) {
    die('Setting MYSQLI_OPT_CONNECT_TIMEOUT failed');
}
if (!mysqli_real_connect($link, 'localhost', 'my_user', 'my_password', 'my_db')) {
    die('Connect Error (' . mysqli_connect_errno() . ') '
            . mysqli_connect_error());
}
echo 'Success... ' . mysqli_get_host_info($link) . "\n";
mysqli_close($link);
?>
```

The above examples will output:

```
Success... MySQL host info: localhost via TCP/IP
```

See Also

mysqli_connect
mysqli_init
mysqli_options
mysqli_ssl_set
mysqli_close

2.7.39. mysqli::real_escape_string, mysqli_real_escape_string

- mysqli::real_escape_string

  mysqli_real_escape_string

  Escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection

Description

Object oriented style

```
string mysqli::escape_string(string escapestr);
```

```
string mysqli::real_escape_string(string escapestr);
```

Procedural style

```
string mysqli_real_escape_string(mysqli link,
                                 string escapestr);
```

This function is used to create a legal SQL string that you can use in an SQL statement. The given string is encoded to an escaped SQL string, taking into account the current character set of the connection.

Parameters

link	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`
escapestr	The string to be escaped. Characters encoded are `NUL (ASCII 0), \n, \r, \, ', ", and Control-Z`.

Return Values

Returns an escaped string.

Examples

Example 2.32. `mysqli::real_escape_string` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TEMPORARY TABLE myCity LIKE City");
$city = "'s Hertogenbosch";
/* this query will fail, cause we didn't escape $city */
if (!$mysqli->query("INSERT into myCity (Name) VALUES ('$city')")) {
    printf("Error: %s\n", $mysqli->sqlstate);
}
$city = $mysqli->real_escape_string($city);
/* this query with escaped $city will work */
if ($mysqli->query("INSERT into myCity (Name) VALUES ('$city')")) {
    printf("%d Row inserted.\n", $mysqli->affected_rows);
}
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TEMPORARY TABLE myCity LIKE City");
$city = "'s Hertogenbosch";
/* this query will fail, cause we didn't escape $city */
if (!mysqli_query($link, "INSERT into myCity (Name) VALUES ('$city')")) {
    printf("Error: %s\n", mysqli_sqlstate($link));
}
```

```
$city = mysqli_real_escape_string($link, $city);
/* this query with escaped $city will work */
if (mysqli_query($link, "INSERT into myCity (Name) VALUES ('$city')")) {
    printf("%d Row inserted.\n", mysqli_affected_rows($link));
}
mysqli_close($link);
?>
```

The above examples will output:

```
Error: 42000
1 Row inserted.
```

See Also

mysqli_character_set_name

2.7.40. mysqli::real_query, mysqli_real_query

- mysqli::real_query

  mysqli_real_query

  Execute an SQL query

Description

Object oriented style

```
bool mysqli::real_query(string query);
```

Procedural style

```
bool mysqli_real_query(mysqli link,
                       string query);
```

Executes a single query against the database whose result can then be retrieved or stored using the mysqli_store_result or mysqli_use_result functions.

In order to determine if a given query should return a result set or not, see mysqli_field_count.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
query	The query, as a string. Data inside the query should be properly escaped.

Return Values

Returns TRUE on success or FALSE on failure.

See Also

mysqli_query
mysqli_store_result
mysqli_use_result

2.7.41. mysqli::reap_async_query, mysqli_reap_async_query

- mysqli::reap_async_query

  mysqli_reap_async_query

  Get result from async query

Description

Object oriented style

```
public mysqli_result mysqli::reap_async_query();
```

Procedural style

```
mysqli_result mysqli_reap_async_query(mysql link);
```

> **Warning**
>
> This function is currently not documented; only its argument list is available.

Get result from async query. Available only with mysqlnd.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

Returns mysqli_result in success, FALSE otherwise.

See Also

mysqli_poll

2.7.42. mysqli::rollback, mysqli_rollback

- mysqli::rollback

  mysqli_rollback

  Rolls back current transaction

Description

Object oriented style

```
bool mysqli::rollback();
```

Procedural style

```
bool mysqli_rollback(mysqli link);
```

Rollbacks the current transaction for the database.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

Returns TRUE on success or FALSE on failure.

Examples

Example 2.33. mysqli::rollback example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* disable autocommit */
$mysqli->autocommit(FALSE);
$mysqli->query("CREATE TABLE myCity LIKE City");
$mysqli->query("ALTER TABLE myCity Type=InnoDB");
$mysqli->query("INSERT INTO myCity SELECT * FROM City LIMIT 50");
/* commit insert */
$mysqli->commit();
/* delete all rows */
$mysqli->query("DELETE FROM myCity");
if ($result = $mysqli->query("SELECT COUNT(*) FROM myCity")) {
    $row = $result->fetch_row();
    printf("%d rows in table myCity.\n", $row[0]);
    /* Free result */
    $result->close();
}
/* Rollback */
$mysqli->rollback();
if ($result = $mysqli->query("SELECT COUNT(*) FROM myCity")) {
    $row = $result->fetch_row();
    printf("%d rows in table myCity (after rollback).\n", $row[0]);
    /* Free result */
    $result->close();
}
/* Drop table myCity */
$mysqli->query("DROP TABLE myCity");
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* disable autocommit */
```

```
mysqli_autocommit($link, FALSE);
mysqli_query($link, "CREATE TABLE myCity LIKE City");
mysqli_query($link, "ALTER TABLE myCity Type=InnoDB");
mysqli_query($link, "INSERT INTO myCity SELECT * FROM City LIMIT 50");
/* commit insert */
mysqli_commit($link);
/* delete all rows */
mysqli_query($link, "DELETE FROM myCity");
if ($result = mysqli_query($link, "SELECT COUNT(*) FROM myCity")) {
    $row = mysqli_fetch_row($result);
    printf("%d rows in table myCity.\n", $row[0]);
    /* Free result */
    mysqli_free_result($result);
}
/* Rollback */
mysqli_rollback($link);
if ($result = mysqli_query($link, "SELECT COUNT(*) FROM myCity")) {
    $row = mysqli_fetch_row($result);
    printf("%d rows in table myCity (after rollback).\n", $row[0]);
    /* Free result */
    mysqli_free_result($result);
}
/* Drop table myCity */
mysqli_query($link, "DROP TABLE myCity");
mysqli_close($link);
?>
```

The above examples will output:

```
0 rows in table myCity.
50 rows in table myCity (after rollback).
```

See Also

mysqli_commit
mysqli_autocommit

2.7.43. mysqli::select_db, mysqli_select_db

- mysqli::select_db

  mysqli_select_db

  Selects the default database for database queries

Description

Object oriented style

```
bool mysqli::select_db(string dbname);
```

Procedural style

```
bool mysqli_select_db(mysqli link,
                      string dbname);
```

Selects the default database to be used when performing queries against the database connection.

> **Note**
>
> This function should only be used to change the default database for the connection. You can select the default database

with 4th parameter in mysqli_connect.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
dbname	The database name.

Return Values

Returns TRUE on success or FALSE on failure.

Examples

Example 2.34. mysqli::select_db example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* return name of current default database */
if ($result = $mysqli->query("SELECT DATABASE()")) {
    $row = $result->fetch_row();
    printf("Default database is %s.\n", $row[0]);
    $result->close();
}
/* change db to world db */
$mysqli->select_db("world");
/* return name of current default database */
if ($result = $mysqli->query("SELECT DATABASE()")) {
    $row = $result->fetch_row();
    printf("Default database is %s.\n", $row[0]);
    $result->close();
}
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "test");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* return name of current default database */
if ($result = mysqli_query($link, "SELECT DATABASE()")) {
    $row = mysqli_fetch_row($result);
    printf("Default database is %s.\n", $row[0]);
    mysqli_free_result($result);
}
/* change db to world db */
mysqli_select_db($link, "world");
/* return name of current default database */
if ($result = mysqli_query($link, "SELECT DATABASE()")) {
    $row = mysqli_fetch_row($result);
    printf("Default database is %s.\n", $row[0]);
    mysqli_free_result($result);
}
mysqli_close($link);
?>
```

The above examples will output:

```
Default database is test.
Default database is world.
```

See Also

mysqli_connect
mysqli_real_connect

2.7.44. mysqli::set_charset, mysqli_set_charset

- mysqli::set_charset

  mysqli_set_charset

  Sets the default client character set

Description

Object oriented style

```
bool mysqli::set_charset(string charset);
```

Procedural style

```
bool mysqli_set_charset(mysqli link,
                        string charset);
```

Sets the default character set to be used when sending data from and to the database server.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
charset	The charset to be set as default.

Return Values

Returns TRUE on success or FALSE on failure.

Notes

> **Note**
>
> To use this function on a Windows platform you need MySQL client library version 4.1.11 or above (for MySQL 5.0 you need 5.0.6 or above).

> **Note**
>
> This is the preferred way to change the charset. Using mysqli::query to execute SET NAMES .. is not recommended.

Examples

Example 2.35. `mysqli::set_charset` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* change character set to utf8 */
if (!$mysqli->set_charset("utf8")) {
    printf("Error loading character set utf8: %s\n", $mysqli->error);
} else {
    printf("Current character set: %s\n", $mysqli->character_set_name());
}
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect('localhost', 'my_user', 'my_password', 'test');
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* change character set to utf8 */
if (!mysqli_set_charset($link, "utf8")) {
    printf("Error loading character set utf8: %s\n", mysqli_error($link));
} else {
    printf("Current character set: %s\n", mysqli_character_set_name($link));
}
mysqli_close($link);
?>
```

The above examples will output:

```
Current character set: utf8
```

See Also

`mysqli_character_set_name`
`mysqli_real_escape_string`
List of character sets that MySQL supports

2.7.45. `mysqli::set_local_infile_default`, `mysqli_set_local_infile_default`

- `mysqli::set_local_infile_default`

  `mysqli_set_local_infile_default`

Unsets user defined handler for load local infile command

Description

```
void mysqli_set_local_infile_default(mysqli link);
```

Deactivates a `LOAD DATA INFILE LOCAL` handler previously set with `mysqli_set_local_infile_handler`.

Parameters

`link`	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

No value is returned.

Examples

See `mysqli_set_local_infile_handler` examples

See Also

`mysqli_set_local_infile_handler`

2.7.46. `mysqli::set_local_infile_handler`, `mysqli_set_local_infile_handler`

- `mysqli::set_local_infile_handler`

  `mysqli_set_local_infile_handler`

  Set callback function for LOAD DATA LOCAL INFILE command

Description

Object oriented style

```
bool mysqli::set_local_infile_handler(mysqli link,
                                      callback read_func);
```

Procedural style

```
bool mysqli_set_local_infile_handler(mysqli link,
                                     callback read_func);
```

Set callback function for LOAD DATA LOCAL INFILE command

The callbacks task is to read input from the file specified in the `LOAD DATA LOCAL INFILE` and to reformat it into the format understood by `LOAD DATA INFILE`.

The returned data needs to match the format specified in the `LOAD DATA`

Parameters

`link`	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`
`read_func`	A callback function or object method taking the following parameters:

`stream`	A PHP stream associated with the SQL commands INFILE
`&buffer`	A string buffer to store the rewritten input into
`buflen`	The maximum number of characters to be stored in the buffer
`&errormsg`	If an error occurs you can store an error message in here

The callback function should return the number of characters stored in the *`buffer`* or a negative value if an error occurred.

Return Values

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

Examples

Example 2.36. `mysqli::set_local_infile_handler` example

Object oriented style

```
<?php
  $db = mysqli_init();
  $db->real_connect("localhost","root","","test");
  function callme($stream, &$buffer, $buflen, &$errmsg)
  {
    $buffer = fgets($stream);
    echo $buffer;
    // convert to upper case and replace "," delimiter with [TAB]
    $buffer = strtoupper(str_replace(",", "\t", $buffer));
    return strlen($buffer);
  }
  echo "Input:\n";
  $db->set_local_infile_handler("callme");
  $db->query("LOAD DATA LOCAL INFILE 'input.txt' INTO TABLE t1");
  $db->set_local_infile_default();
  $res = $db->query("SELECT * FROM t1");
  echo "\nResult:\n";
  while ($row = $res->fetch_assoc()) {
    echo join(",", $row)."\n";
  }
?>
```

Procedural style

```
<?php
  $db = mysqli_init();
  mysqli_real_connect($db, "localhost","root","","test");
  function callme($stream, &$buffer, $buflen, &$errmsg)
  {
    $buffer = fgets($stream);
    echo $buffer;
    // convert to upper case and replace "," delimiter with [TAB]
    $buffer = strtoupper(str_replace(",", "\t", $buffer));
    return strlen($buffer);
  }
  echo "Input:\n";
  mysqli_set_local_infile_handler($db, "callme");
  mysqli_query($db, "LOAD DATA LOCAL INFILE 'input.txt' INTO TABLE t1");
  mysqli_set_local_infile_default($db);
  $res = mysqli_query($db, "SELECT * FROM t1");
  echo "\nResult:\n";
  while ($row = mysqli_fetch_assoc($res)) {
    echo join(",", $row)."\n";
  }
?>
```

The above examples will output:

```
Input:
23,foo
42,bar
Output:
23,FOO
42,BAR
```

See Also

mysqli_set_local_infile_default

2.7.47. mysqli->sqlstate, mysqli_sqlstate

- mysqli->sqlstate

  mysqli_sqlstate

  Returns the SQLSTATE error from previous MySQL operation

Description

Object oriented style

```
 mysqli {

  string sqlstate ;

}
```

Procedural style

```
  string mysqli_sqlstate(mysqli link);
```

Returns a string containing the SQLSTATE error code for the last error. The error code consists of five characters. `'00000'` means no error. The values are specified by ANSI SQL and ODBC. For a list of possible values, see http://dev.mysql.com/doc/mysql/en/error-handling.html.

> **Note**
>
> Note that not all MySQL errors are yet mapped to SQLSTATE's. The value `HY000` (general error) is used for unmapped errors.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

Returns a string containing the SQLSTATE error code for the last error. The error code consists of five characters. `'00000'` means no error.

Examples

Example 2.37. mysqli->sqlstate example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* Table City already exists, so we should get an error */
if (!$mysqli->query("CREATE TABLE City (ID INT, Name VARCHAR(30))")) {
    printf("Error - SQLSTATE %s.\n", $mysqli->sqlstate);
}
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* Table City already exists, so we should get an error */
if (!mysqli_query($link, "CREATE TABLE City (ID INT, Name VARCHAR(30))")) {
    printf("Error - SQLSTATE %s.\n", mysqli_sqlstate($link));
}
mysqli_close($link);
?>
```

The above examples will output:

```
Error - SQLSTATE 42S01.
```

See Also

mysqli_errno
mysqli_error

2.7.48. mysqli::ssl_set, mysqli_ssl_set

- mysqli::ssl_set

  mysqli_ssl_set

  Used for establishing secure connections using SSL

Description

Object oriented style

```
  bool mysqli::ssl_set(string key,
                       string cert,
                       string ca,
                       string capath,
                       string cipher);
```

Procedural style

```
bool mysqli_ssl_set(mysqli link,
                    string key,
                    string cert,
                    string ca,
                    string capath,
                    string cipher);
```

Used for establishing secure connections using SSL. It must be called before `mysqli_real_connect`. This function does nothing unless OpenSSL support is enabled.

Note that MySQL Native Driver does not support SSL, so calling this function when using MySQL Native Driver will result in an error. MySQL Native Driver is enabled by default on Microsoft Windows from PHP version 5.3 onwards.

Parameters

`link`	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`
`key`	The path name to the key file.
`cert`	The path name to the certificate file.
`ca`	The path name to the certificate authority file.
`capath`	The pathname to a directory that contains trusted SSL CA certificates in PEM format.
`cipher`	A list of allowable ciphers to use for SSL encryption.

Any unused SSL parameters may be given as `NULL`

Return Values

This function always returns `TRUE` value. If SSL setup is incorrect `mysqli_real_connect` will return an error when you attempt to connect.

See Also

`mysqli_options`
`mysqli_real_connect`

2.7.49. `mysqli::stat`, `mysqli_stat`

- `mysqli::stat`

  `mysqli_stat`

  Gets the current system status

Description

Object oriented style

```
string mysqli::stat();
```

Procedural style

```
string mysqli_stat(mysqli link);
```

mysqli_stat returns a string containing information similar to that provided by the 'mysqladmin status' command. This includes uptime in seconds and the number of running threads, questions, reloads, and open tables.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

A string describing the server status. FALSE if an error occurred.

Examples

Example 2.38. mysqli::stat example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
printf ("System status: %s\n", $mysqli->stat());
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
printf("System status: %s\n", mysqli_stat($link));
mysqli_close($link);
?>
```

The above examples will output:

```
System status: Uptime: 272  Threads: 1  Questions: 5340  Slow queries: 0
Opens: 13  Flush tables: 1  Open tables: 0  Queries per second avg: 19.632
Memory in use: 8496K  Max memory used: 8560K
```

See Also

mysqli_get_server_info

2.7.50. mysqli::stmt_init, mysqli_stmt_init

- mysqli::stmt_init

  mysqli_stmt_init

  Initializes a statement and returns an object for use with mysqli_stmt_prepare

Description

Object oriented style

```
mysqli_stmt mysqli::stmt_init();
```

Procedural style

```
mysqli_stmt mysqli_stmt_init(mysqli link);
```

Allocates and initializes a statement object suitable for mysqli_stmt_prepare.

> **Note**
>
> Any subsequent calls to any mysqli_stmt function will fail until mysqli_stmt_prepare was called.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

Returns an object.

See Also

mysqli_stmt_prepare

2.7.51. mysqli::store_result, mysqli_store_result

- mysqli::store_result

  mysqli_store_result

  Transfers a result set from the last query

Description

Object oriented style

```
mysqli_result mysqli::store_result();
```

Procedural style

```
mysqli_result mysqli_store_result(mysqli link);
```

Transfers the result set from the last query on the database connection represented by the *link* parameter to be used with the mysqli_data_seek function.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

Returns a buffered result object or FALSE if an error occurred.

> **Note**
>
> mysqli_store_result returns FALSE in case the query didn't return a result set (if the query was, for example an INSERT statement). This function also returns FALSE if the reading of the result set failed. You can check if you have got an error by checking if mysqli_error doesn't return an empty string, if mysqli_errno returns a non zero value, or if mysqli_field_count returns a non zero value. Also possible reason for this function returning FALSE after successful call to mysqli_query can be too large result set (memory for it cannot be allocated). If mysqli_field_count returns a non-zero value, the statement should have produced a non-empty result set.

Notes

> **Note**
>
> Although it is always good practice to free the memory used by the result of a query using the mysqli_free_result function, when transferring large result sets using the mysqli_store_result this becomes particularly important.

Examples

See mysqli_multi_query.

See Also

mysqli_real_query
mysqli_use_result

2.7.52. mysqli->thread_id, mysqli_thread_id

- mysqli->thread_id

  mysqli_thread_id

  Returns the thread ID for the current connection

Description

Object oriented style

```
mysqli {
  int thread_id ;
}
```

Procedural style

```
  int mysqli_thread_id(mysqli link);
```

The mysqli_thread_id function returns the thread ID for the current connection which can then be killed using the mysqli_kill function. If the connection is lost and you reconnect with mysqli_ping, the thread ID will be other. Therefore you should get the thread ID only when you need it.

> **Note**
>
> The thread ID is assigned on a connection-by-connection basis. Hence, if the connection is broken and then re-established a new thread ID will be assigned.
>
> To kill a running query you can use the SQL command `KILL QUERY processid`.

Parameters

`link`	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

Returns the Thread ID for the current connection.

Examples

Example 2.39. `mysqli->thread_id` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* determine our thread id */
$thread_id = $mysqli->thread_id;
/* Kill connection */
$mysqli->kill($thread_id);
/* This should produce an error */
if (!$mysqli->query("CREATE TABLE myCity LIKE City")) {
    printf("Error: %s\n", $mysqli->error);
    exit;
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* determine our thread id */
$thread_id = mysqli_thread_id($link);
/* Kill connection */
mysqli_kill($link, $thread_id);
/* This should produce an error */
if (!mysqli_query($link, "CREATE TABLE myCity LIKE City")) {
    printf("Error: %s\n", mysqli_error($link));
    exit;
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Error: MySQL server has gone away
```

See Also

mysqli_kill

2.7.53. mysqli::thread_safe, mysqli_thread_safe

- mysqli::thread_safe

  mysqli_thread_safe

  Returns whether thread safety is given or not

Description

Procedural style

```
bool mysqli_thread_safe();
```

Tells whether the client library is compiled as thread-safe.

Return Values

TRUE if the client library is thread-safe, otherwise FALSE .

2.7.54. mysqli::use_result, mysqli_use_result

- mysqli::use_result

  mysqli_use_result

  Initiate a result set retrieval

Description

Object oriented style

```
mysqli_result mysqli::use_result();
```

Procedural style

```
mysqli_result mysqli_use_result(mysqli link);
```

Used to initiate the retrieval of a result set from the last query executed using the mysqli_real_query function on the database connection.

Either this or the mysqli_store_result function must be called before the results of a query can be retrieved, and one or the other must be called to prevent the next query on that database connection from failing.

Note

> The `mysqli_use_result` function does not transfer the entire result set from the database and hence cannot be used functions such as `mysqli_data_seek` to move to a particular row within the set. To use this functionality, the result set must be stored using `mysqli_store_result`. One should not use `mysqli_use_result` if a lot of processing on the client side is performed, since this will tie up the server and prevent other threads from updating any tables from which the data is being fetched.

Return Values

Returns an unbuffered result object or `FALSE` if an error occurred.

Examples

Example 2.40. `mysqli::use_result` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query  = "SELECT CURRENT_USER();";
$query .= "SELECT Name FROM City ORDER BY ID LIMIT 20, 5";
/* execute multi query */
if ($mysqli->multi_query($query)) {
    do {
        /* store first result set */
        if ($result = $mysqli->use_result()) {
            while ($row = $result->fetch_row()) {
                printf("%s\n", $row[0]);
            }
            $result->close();
        }
        /* print divider */
        if ($mysqli->more_results()) {
            printf("-----------------\n");
        }
    } while ($mysqli->next_result());
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query  = "SELECT CURRENT_USER();";
$query .= "SELECT Name FROM City ORDER BY ID LIMIT 20, 5";
/* execute multi query */
if (mysqli_multi_query($link, $query)) {
    do {
        /* store first result set */
        if ($result = mysqli_use_result($link)) {
            while ($row = mysqli_fetch_row($result)) {
                printf("%s\n", $row[0]);
            }
            mysqli_free_result($result);
        }
        /* print divider */
        if (mysqli_more_results($link)) {
            printf("-----------------\n");
        }
    } while (mysqli_next_result($link));
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
my_user@localhost
-----------------
Amersfoort
Maastricht
Dordrecht
Leiden
Haarlemmermeer
```

See Also

mysqli_real_query
mysqli_store_result

2.7.55. mysqli->warning_count, mysqli_warning_count

Copyright 1997-2010 the PHP Documentation Group.

- mysqli->warning_count

  mysqli_warning_count

  Returns the number of warnings from the last query for the given link

Description

Object oriented style

```
 mysqli {
  int warning_count ;
}
```

Procedural style

```
  int mysqli_warning_count(mysqli link);
```

Returns the number of warnings from the last query in the connection.

> **Note**
>
> For retrieving warning messages you can use the SQL command `SHOW WARNINGS [limit row_count]`.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init

Return Values

Number of warnings or zero if there are no warnings.

Examples

Example 2.41. mysqli->warning_count example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TABLE myCity LIKE City");
/* a remarkable city in Wales */
$query = "INSERT INTO myCity (CountryCode, Name) VALUES('GBR',
        'Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch')";
$mysqli->query($query);
if ($mysqli->warning_count) {
    if ($result = $mysqli->query("SHOW WARNINGS")) {
        $row = $result->fetch_row();
        printf("%s (%d): %s\n", $row[0], $row[1], $row[2]);
        $result->close();
    }
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TABLE myCity LIKE City");
/* a remarkable long city name in Wales */
$query = "INSERT INTO myCity (CountryCode, Name) VALUES('GBR',
        'Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch')";
mysqli_query($link, $query);
if (mysqli_warning_count($link)) {
    if ($result = mysqli_query($link, "SHOW WARNINGS")) {
        $row = mysqli_fetch_row($result);
        printf("%s (%d): %s\n", $row[0], $row[1], $row[2]);
        mysqli_free_result($result);
    }
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Warning (1264): Data truncated for column 'Name' at row 1
```

See Also

mysqli_errno
mysqli_error
mysqli_sqlstate

2.8. The MySQLi_STMT class (MySQLi_STMT)

Represents a prepared statement.

```
 MySQLi_STMT {
 MySQLi_STMT
      Properties

  int affected_rows ;


  int errno ;


  string error ;


  int field_count ;


  int insert_id ;


  int num_rows ;


  int param_count ;


  string sqlstate ;

Methods

  int mysqli_stmt_affected_rows(mysqli_stmt stmt);


  int mysqli_stmt::attr_get(int attr);


  bool mysqli_stmt::attr_set(int attr,
                             int mode);


  bool mysqli_stmt::bind_param(string types,
                               mixed var1,
                               mixed ...);


  bool mysqli_stmt::bind_result(mixed var1,
                                mixed ...);


  bool mysqli_stmt::close();


  void mysqli_stmt::data_seek(int offset);


  int mysqli_stmt_errno(mysqli_stmt stmt);


  string mysqli_stmt_error(mysqli_stmt stmt);


  bool mysqli_stmt::execute();


  bool mysqli_stmt::fetch();


  int mysqli_stmt_field_count(mysqli_stmt stmt);


  void mysqli_stmt::free_result();


  object mysqli_stmt::get_warnings(mysqli_stmt stmt);


  mixed mysqli_stmt_insert_id(mysqli_stmt stmt);
  int mysqli_stmt_num_rows(mysqli_stmt stmt);

  int mysqli_stmt_param_count(mysqli_stmt stmt);

  mixed mysqli_stmt::prepare(string query);

  bool mysqli_stmt::reset();

  mysqli_result mysqli_stmt::result_metadata();

  bool mysqli_stmt::send_long_data(int param_nr,
                                   string data);

  string mysqli_stmt_sqlstate(mysqli_stmt stmt);

  bool mysqli_stmt::store_result();
}
```

2.8.1. mysqli_stmt->affected_rows, mysqli_stmt_affected_rows

- mysqli_stmt->affected_rows

  mysqli_stmt_affected_rows

  Returns the total number of rows changed, deleted, or inserted by the last executed statement

Description

Object oriented style

```
 mysqli_stmt {
  int affected_rows ;
}
```

Procedural style

```
  int mysqli_stmt_affected_rows(mysqli_stmt stmt);
```

Returns the number of rows affected by `INSERT`, `UPDATE`, or `DELETE` query.

This function only works with queries which update a table. In order to get the number of rows from a SELECT query, use `mysqli_stmt_num_rows` instead.

Parameters

stmt	Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records where updated for an UPDATE/DELETE statement, no rows matched the WHERE clause in the query or that no query has yet been executed. -1 indicates that the query has returned an error. NULL indicates an invalid argument was supplied to the function.

Note

If the number of affected rows is greater than maximal PHP int value, the number of affected rows will be returned as a string value.

Examples

Example 2.42. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* create temp table */
$mysqli->query("CREATE TEMPORARY TABLE myCountry LIKE Country");
$query = "INSERT INTO myCountry SELECT * FROM Country WHERE Code LIKE ?";
/* prepare statement */
if ($stmt = $mysqli->prepare($query)) {
    /* Bind variable for placeholder */
    $code = 'A%';
    $stmt->bind_param("s", $code);
    /* execute statement */
    $stmt->execute();
    printf("rows inserted: %d\n", $stmt->affected_rows);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 2.43. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* create temp table */
mysqli_query($link, "CREATE TEMPORARY TABLE myCountry LIKE Country");
$query = "INSERT INTO myCountry SELECT * FROM Country WHERE Code LIKE ?";
/* prepare statement */
if ($stmt = mysqli_prepare($link, $query)) {
    /* Bind variable for placeholder */
    $code = 'A%';
    mysqli_stmt_bind_param($stmt, "s", $code);
    /* execute statement */
    mysqli_stmt_execute($stmt);
    printf("rows inserted: %d\n", mysqli_stmt_affected_rows($stmt));
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
rows inserted: 17
```

See Also

mysqli_stmt_num_rows
mysqli_prepare

2.8.2. mysqli_stmt::attr_get, mysqli_stmt_attr_get

- mysqli_stmt::attr_get

  mysqli_stmt_attr_get

  Used to get the current value of a statement attribute

Description

Object oriented style

```
int mysqli_stmt::attr_get(int attr);
```

Procedural style

```
int mysqli_stmt_attr_get(mysqli_stmt stmt,
                         int attr);
```

Gets the current value of a statement attribute.

Parameters

stmt	Procedural style only: A statement identifier returned by mysqli_stmt_init.
attr	The attribute that you want to get.

Return Values

Returns FALSE if the attribute is not found, otherwise returns the value of the attribute.

2.8.3. mysqli_stmt::attr_set, mysqli_stmt_attr_set

- mysqli_stmt::attr_set

  mysqli_stmt_attr_set

  Used to modify the behavior of a prepared statement

Description

Object oriented style

```
bool mysqli_stmt::attr_set(int attr,
                           int mode);
```

Procedural style

```
bool mysqli_stmt_attr_set(mysqli_stmt stmt,
                             int attr,
                             int mode);
```

Used to modify the behavior of a prepared statement. This function may be called multiple times to set several attributes.

Parameters

stmt	Procedural style only: A statement identifier returned by mysqli_stmt_init.
attr	The attribute that you want to set. It can have one of the following values:

Table 2.5. Attribute values

Character	Description
MYSQLI_STMT_ATTR_UPDATE_MAX_LENGTH	If set to 1, causes mysqli_stmt_store_result to update the metadata MYSQL_FIELD->max_length value.
MYSQLI_STMT_ATTR_CURSOR_TYPE	Type of cursor to open for statement when mysqli_stmt_execute is invoked. *mode* can be MYSQLI_CURSOR_TYPE_NO_CURSOR (the default) or MYSQLI_CURSOR_TYPE_READ_ONLY.
MYSQLI_STMT_ATTR_PREFETCH_ROWS	Number of rows to fetch from server at a time when using a cursor. *mode* can be in the range from 1 to the maximum value of unsigned long. The default is 1.

If you use the MYSQLI_STMT_ATTR_CURSOR_TYPE option with MYSQLI_CURSOR_TYPE_READ_ONLY, a cursor is opened for the statement when you invoke mysqli_stmt_execute. If there is already an open cursor from a previous mysqli_stmt_execute call, it closes the cursor before opening a new one. mysqli_stmt_reset also closes any open cursor before preparing the statement for re-execution. mysqli_stmt_free_result closes any open cursor.

If you open a cursor for a prepared statement, mysqli_stmt_store_result is unnecessary.

mode	The value to assign to the attribute.

2.8.4. mysqli_stmt::bind_param, mysqli_stmt_bind_param

- mysqli_stmt::bind_param

  mysqli_stmt_bind_param

  Binds variables to a prepared statement as parameters

Description

Object oriented style

```
bool mysqli_stmt::bind_param(string types,
                             mixed var1,
                             mixed ...);
```

Procedural style

```
bool mysqli_stmt_bind_param(mysqli_stmt stmt,
                            string types,
                            mixed var1,
                            mixed ...);
```

Bind variables for the parameter markers in the SQL statement that was passed to `mysqli_prepare`.

> **Note**
>
> If data size of a variable exceeds max. allowed packet size (max_allowed_packet), you have to specify `b` in *`types`* and use `mysqli_stmt_send_long_data` to send the data in packets.

> **Note**
>
> Care must be taken when using `mysqli_stmt_bind_param` in conjunction with `call_user_func_array`. Note that `mysqli_stmt_bind_param` requires parameters to be passed by reference, whereas `call_user_func_array` can accept as a parameter a list of variables that can represent references or values.

Parameters

`stmt` Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

`types` A string that contains one or more characters which specify the types for the corresponding bind variables:

Table 2.6. Type specification chars

Character	Description
i	corresponding variable has type integer
d	corresponding variable has type double
s	corresponding variable has type string
b	corresponding variable is a blob and will be sent in packets

`var1` The number of variables and length of string *`types`* must match the parameters in the statement.

Return Values

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

Examples

Example 2.44. Object oriented style

```
<?php
$mysqli = new mysqli('localhost', 'my_user', 'my_password', 'world');
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$stmt = $mysqli->prepare("INSERT INTO CountryLanguage VALUES (?, ?, ?, ?)");
$stmt->bind_param('sssd', $code, $language, $official, $percent);
$code = 'DEU';
$language = 'Bavarian';
$official = "F";
$percent = 11.2;
/* execute prepared statement */
$stmt->execute();
printf("%d Row inserted.\n", $stmt->affected_rows);
/* close statement and connection */
$stmt->close();
/* Clean up table CountryLanguage */
$mysqli->query("DELETE FROM CountryLanguage WHERE Language='Bavarian'");
printf("%d Row deleted.\n", $mysqli->affected_rows);
/* close connection */
$mysqli->close();
?>
```

Example 2.45. Procedural style

```
<?php
$link = mysqli_connect('localhost', 'my_user', 'my_password', 'world');
/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$stmt = mysqli_prepare($link, "INSERT INTO CountryLanguage VALUES (?, ?, ?, ?)");
mysqli_stmt_bind_param($stmt, 'sssd', $code, $language, $official, $percent);
$code = 'DEU';
$language = 'Bavarian';
$official = "F";
$percent = 11.2;
/* execute prepared statement */
mysqli_stmt_execute($stmt);
printf("%d Row inserted.\n", mysqli_stmt_affected_rows($stmt));
/* close statement and connection */
mysqli_stmt_close($stmt);
/* Clean up table CountryLanguage */
mysqli_query($link, "DELETE FROM CountryLanguage WHERE Language='Bavarian'");
printf("%d Row deleted.\n", mysqli_affected_rows($link));
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
1 Row inserted.
1 Row deleted.
```

See Also

mysqli_stmt_bind_result
mysqli_stmt_execute
mysqli_stmt_fetch
mysqli_prepare
mysqli_stmt_send_long_data
mysqli_stmt_errno
mysqli_stmt_error

2.8.5. mysqli_stmt::bind_result, mysqli_stmt_bind_result

- mysqli_stmt::bind_result

  mysqli_stmt_bind_result

  Binds variables to a prepared statement for result storage

Description

Object oriented style

```
  bool mysqli_stmt::bind_result(mixed var1,
                                mixed ...);
```

Procedural style

```
  bool mysqli_stmt_bind_result(mysqli_stmt stmt,
                               mixed var1,
                               mixed ...);
```

Binds columns in the result set to variables.

When `mysqli_stmt_fetch` is called to fetch data, the MySQL client/server protocol places the data for the bound columns into the specified variables *`var1, ...`*.

> **Note**
>
> Note that all columns must be bound after `mysqli_stmt_execute` and prior to calling `mysqli_stmt_fetch`. Depending on column types bound variables can silently change to the corresponding PHP type.
>
> A column can be bound or rebound at any time, even after a result set has been partially retrieved. The new binding takes effect the next time `mysqli_stmt_fetch` is called.

Parameters

`stmt`	Procedural style only: A statement identifier returned by `mysqli_stmt_init`.
`var1`	The variable to be bound.

Return Values

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

Examples

Example 2.46. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* prepare statement */
if ($stmt = $mysqli->prepare("SELECT Code, Name FROM Country ORDER BY Name LIMIT 5")) {
    $stmt->execute();
    /* bind variables to prepared statement */
    $stmt->bind_result($col1, $col2);
    /* fetch values */
    while ($stmt->fetch()) {
        printf("%s %s\n", $col1, $col2);
    }
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 2.47. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* prepare statement */
if ($stmt = mysqli_prepare($link, "SELECT Code, Name FROM Country ORDER BY Name LIMIT 5")) {
    mysqli_stmt_execute($stmt);
    /* bind variables to prepared statement */
    mysqli_stmt_bind_result($stmt, $col1, $col2);
    /* fetch values */
    while (mysqli_stmt_fetch($stmt)) {
        printf("%s %s\n", $col1, $col2);
    }
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
AFG Afghanistan
ALB Albania
DZA Algeria
ASM American Samoa
AND Andorra
```

See Also

mysqli_stmt_bind_param
mysqli_stmt_execute
mysqli_stmt_fetch
mysqli_prepare
mysqli_stmt_prepare
mysqli_stmt_init
mysqli_stmt_errno
mysqli_stmt_error

2.8.6. mysqli_stmt::close, mysqli_stmt_close

- mysqli_stmt::close

  mysqli_stmt_close

  Closes a prepared statement

Description

Object oriented style

```
bool mysqli_stmt::close();
```

Procedural style

```
bool mysqli_stmt_close(mysqli_stmt stmt);
```

Closes a prepared statement. mysqli_stmt_close also deallocates the statement handle. If the current statement has pending or unread results, this function cancels them so that the next query can be executed.

Parameters

stmt	Procedural style only: A statement identifier returned by mysqli_stmt_init.

Return Values

Returns TRUE on success or FALSE on failure.

See Also

mysqli_prepare

2.8.7. mysqli_stmt::data_seek, mysqli_stmt_data_seek

- mysqli_stmt::data_seek

  mysqli_stmt_data_seek

  Seeks to an arbitrary row in statement result set

Description

Object oriented style

```
void mysqli_stmt::data_seek(int offset);
```

Procedural style

```
void mysqli_stmt_data_seek(mysqli_stmt stmt,
                           int offset);
```

Seeks to an arbitrary result pointer in the statement result set.

mysqli_stmt_store_result must be called prior to mysqli_stmt_data_seek.

Parameters

stmt	Procedural style only: A statement identifier returned by mysqli_stmt_init.
offset	Must be between zero and the total number of rows minus one (0.. mysqli_stmt_num_rows - 1).

Return Values

No value is returned.

Examples

Example 2.48. Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name";
```

```
if ($stmt = $mysqli->prepare($query)) {
    /* execute query */
    $stmt->execute();
    /* bind result variables */
    $stmt->bind_result($name, $code);
    /* store result */
    $stmt->store_result();
    /* seek to row no. 400 */
    $stmt->data_seek(399);
    /* fetch values */
    $stmt->fetch();
    printf ("City: %s  Countrycode: %s\n", $name, $code);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 2.49. Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name";
if ($stmt = mysqli_prepare($link, $query)) {
    /* execute query */
    mysqli_stmt_execute($stmt);
    /* bind result variables */
    mysqli_stmt_bind_result($stmt, $name, $code);
    /* store result */
    mysqli_stmt_store_result($stmt);
    /* seek to row no. 400 */
    mysqli_stmt_data_seek($stmt, 399);
    /* fetch values */
    mysqli_stmt_fetch($stmt);
    printf ("City: %s  Countrycode: %s\n", $name, $code);
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
City: Benin City  Countrycode: NGA
```

See Also

mysqli_prepare

2.8.8. mysqli_stmt->errno, mysqli_stmt_errno

- mysqli_stmt->errno

  mysqli_stmt_errno

Returns the error code for the most recent statement call

Description

Object oriented style

```
mysqli_stmt {

  int errno ;

}
```

Procedural style

```
  int mysqli_stmt_errno(mysqli_stmt stmt);
```

Returns the error code for the most recently invoked statement function that can succeed or fail.

Client error message numbers are listed in the MySQL `errmsg.h` header file, server error message numbers are listed in `mysqld_error.h`. In the MySQL source distribution you can find a complete list of error messages and error numbers in the file `Docs/mysqld_error.txt`.

Parameters

`stmt`	Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

An error code value. Zero means no error occurred.

Examples

Example 2.50. Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TABLE myCountry LIKE Country");
$mysqli->query("INSERT INTO myCountry SELECT * FROM Country");
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = $mysqli->prepare($query)) {
    /* drop table */
    $mysqli->query("DROP TABLE myCountry");
    /* execute query */
    $stmt->execute();
    printf("Error: %d.\n", $stmt->errno);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 2.51. Procedural style

```
<?php
```

```
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TABLE myCountry LIKE Country");
mysqli_query($link, "INSERT INTO myCountry SELECT * FROM Country");
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = mysqli_prepare($link, $query)) {
    /* drop table */
    mysqli_query($link, "DROP TABLE myCountry");
    /* execute query */
    mysqli_stmt_execute($stmt);
    printf("Error: %d.\n", mysqli_stmt_errno($stmt));
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Error: 1146.
```

See Also

mysqli_stmt_error
mysqli_stmt_sqlstate

2.8.9. mysqli_stmt->error, mysqli_stmt_error

- mysqli_stmt->error

  mysqli_stmt_error

  Returns a string description for last statement error

Description

Object oriented style

```
mysqli_stmt {
  string error ;
}
```

Procedural style

```
  string mysqli_stmt_error(mysqli_stmt stmt);
```

Returns a containing the error message for the most recently invoked statement function that can succeed or fail.

Parameters

stmt	Procedural style only: A statement identifier returned by mysqli_stmt_init.

Return Values

A string that describes the error. An empty string if no error occurred.

Examples

Example 2.52. Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TABLE myCountry LIKE Country");
$mysqli->query("INSERT INTO myCountry SELECT * FROM Country");
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = $mysqli->prepare($query)) {
    /* drop table */
    $mysqli->query("DROP TABLE myCountry");
    /* execute query */
    $stmt->execute();
    printf("Error: %s.\n", $stmt->error);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 2.53. Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TABLE myCountry LIKE Country");
mysqli_query($link, "INSERT INTO myCountry SELECT * FROM Country");
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = mysqli_prepare($link, $query)) {
    /* drop table */
    mysqli_query($link, "DROP TABLE myCountry");
    /* execute query */
    mysqli_stmt_execute($stmt);
    printf("Error: %s.\n", mysqli_stmt_error($stmt));
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Error: Table 'world.myCountry' doesn't exist.
```

See Also

mysqli_stmt_errno
mysqli_stmt_sqlstate

2.8.10. mysqli_stmt::execute, mysqli_stmt_execute

- mysqli_stmt::execute

  mysqli_stmt_execute

  Executes a prepared Query

Description

Object oriented style

```
bool mysqli_stmt::execute();
```

Procedural style

```
bool mysqli_stmt_execute(mysqli_stmt stmt);
```

Executes a query that has been previously prepared using the `mysqli_prepare` function. When executed any parameter markers which exist will automatically be replaced with the appropriate data.

If the statement is `UPDATE`, `DELETE`, or `INSERT`, the total number of affected rows can be determined by using the `mysqli_stmt_affected_rows` function. Likewise, if the query yields a result set the `mysqli_stmt_fetch` function is used.

> **Note**
>
> When using `mysqli_stmt_execute`, the `mysqli_stmt_fetch` function must be used to fetch the data prior to performing any additional queries.

Parameters

stmt	Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

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

Examples

Example 2.54. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TABLE myCity LIKE City");
/* Prepare an insert statement */
$query = "INSERT INTO myCity (Name, CountryCode, District) VALUES (?,?,?)";
$stmt = $mysqli->prepare($query);
$stmt->bind_param("sss", $val1, $val2, $val3);
$val1 = 'Stuttgart';
```

```
$val2 = 'DEU';
$val3 = 'Baden-Wuerttemberg';
/* Execute the statement */
$stmt->execute();
$val1 = 'Bordeaux';
$val2 = 'FRA';
$val3 = 'Aquitaine';
/* Execute the statement */
$stmt->execute();
/* close statement */
$stmt->close();
/* retrieve all rows from myCity */
$query = "SELECT Name, CountryCode, District FROM myCity";
if ($result = $mysqli->query($query)) {
    while ($row = $result->fetch_row()) {
        printf("%s (%s,%s)\n", $row[0], $row[1], $row[2]);
    }
    /* free result set */
    $result->close();
}
/* remove table */
$mysqli->query("DROP TABLE myCity");
/* close connection */
$mysqli->close();
?>
```

Example 2.55. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TABLE myCity LIKE City");
/* Prepare an insert statement */
$query = "INSERT INTO myCity (Name, CountryCode, District) VALUES (?,?,?)";
$stmt = mysqli_prepare($link, $query);
mysqli_stmt_bind_param($stmt, "sss", $val1, $val2, $val3);
$val1 = 'Stuttgart';
$val2 = 'DEU';
$val3 = 'Baden-Wuerttemberg';
/* Execute the statement */
mysqli_stmt_execute($stmt);
$val1 = 'Bordeaux';
$val2 = 'FRA';
$val3 = 'Aquitaine';
/* Execute the statement */
mysqli_stmt_execute($stmt);
/* close statement */
mysqli_stmt_close($stmt);
/* retrieve all rows from myCity */
$query = "SELECT Name, CountryCode, District FROM myCity";
if ($result = mysqli_query($link, $query)) {
    while ($row = mysqli_fetch_row($result)) {
        printf("%s (%s,%s)\n", $row[0], $row[1], $row[2]);
    }
    /* free result set */
    mysqli_free_result($result);
}
/* remove table */
mysqli_query($link, "DROP TABLE myCity");
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Stuttgart (DEU,Baden-Wuerttemberg)
Bordeaux (FRA,Aquitaine)
```

See Also

mysqli_prepare
mysqli_stmt_bind_param

2.8.11. mysqli_stmt::fetch, mysqli_stmt_fetch

- mysqli_stmt::fetch

  mysqli_stmt_fetch

  Fetch results from a prepared statement into the bound variables

Description

Object oriented style

```
bool mysqli_stmt::fetch();
```

Procedural style

```
bool mysqli_stmt_fetch(mysqli_stmt stmt);
```

Fetch the result from a prepared statement into the variables bound by mysqli_stmt_bind_result.

> **Note**
>
> Note that all columns must be bound by the application before calling mysqli_stmt_fetch.

> **Note**
>
> Data are transferred unbuffered without calling mysqli_stmt_store_result which can decrease performance (but reduces memory cost).

Parameters

stmt	Procedural style only: A statement identifier returned by mysqli_stmt_init.

Return Values

Table 2.7. Return Values

Value	Description
TRUE	Success. Data has been fetched
FALSE	Error occurred
NULL	No more rows/data exists or data truncation occurred

Examples

Example 2.56. Object oriented style

```
<?php
```

```
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 150,5";
if ($stmt = $mysqli->prepare($query)) {
    /* execute statement */
    $stmt->execute();
    /* bind result variables */
    $stmt->bind_result($name, $code);
    /* fetch values */
    while ($stmt->fetch()) {
        printf ("%s (%s)\n", $name, $code);
    }
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 2.57. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 150,5";
if ($stmt = mysqli_prepare($link, $query)) {
    /* execute statement */
    mysqli_stmt_execute($stmt);
    /* bind result variables */
    mysqli_stmt_bind_result($stmt, $name, $code);
    /* fetch values */
    while (mysqli_stmt_fetch($stmt)) {
        printf ("%s (%s)\n", $name, $code);
    }
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Rockford (USA)
Tallahassee (USA)
Salinas (USA)
Santa Clarita (USA)
Springfield (USA)
```

See Also

mysqli_prepare
mysqli_stmt_errno
mysqli_stmt_error
mysqli_stmt_bind_result

2.8.12. mysqli_stmt->field_count, mysqli_stmt_field_count

- mysqli_stmt->field_count

  mysqli_stmt_field_count

  Returns the number of field in the given statement

Description

Object oriented style

```
mysqli_stmt {
  int field_count ;
}
```

Procedural style

```
int mysqli_stmt_field_count(mysqli_stmt stmt);
```

Warning

This function is currently not documented; only its argument list is available.

2.8.13. mysqli_stmt::free_result, mysqli_stmt_free_result

- mysqli_stmt::free_result

  mysqli_stmt_free_result

  Frees stored result memory for the given statement handle

Description

Object oriented style

```
void mysqli_stmt::free_result();
```

Procedural style

```
void mysqli_stmt_free_result(mysqli_stmt stmt);
```

Frees the result memory associated with the statement, which was allocated by mysqli_stmt_store_result.

Parameters

stmt	Procedural style only: A statement identifier returned by mysqli_stmt_init.

Return Values

No value is returned.

See Also

mysqli_stmt_store_result

2.8.14. mysqli_stmt::get_warnings, mysqli_stmt_get_warnings

- mysqli_stmt::get_warnings

  mysqli_stmt_get_warnings

  Get result of SHOW WARNINGS

Description

Object oriented style

```
object mysqli_stmt::get_warnings(mysqli_stmt stmt);
```

Procedural style

```
object mysqli_stmt_get_warnings(mysqli_stmt stmt);
```

> **Warning**
>
> This function is currently not documented; only its argument list is available.

2.8.15. mysqli_stmt->insert_id, mysqli_stmt_insert_id

- mysqli_stmt->insert_id

  mysqli_stmt_insert_id

  Get the ID generated from the previous INSERT operation

Description

Object oriented style

```
mysqli_stmt {
  int insert_id ;
}
```

Procedural style

```
mixed mysqli_stmt_insert_id(mysqli_stmt stmt);
```

> **Warning**
>
> This function is currently not documented; only its argument list is available.

2.8.16. mysqli_stmt::num_rows, mysqli_stmt_num_rows

- `mysqli_stmt::num_rows`

  `mysqli_stmt_num_rows`

  Return the number of rows in statements result set

Description

Object oriented style

```
mysqli_stmt {

  int num_rows ;

}
```

Procedural style

```
  int mysqli_stmt_num_rows(mysqli_stmt stmt);
```

Returns the number of rows in the result set. The use of `mysqli_stmt_num_rows` depends on whether or not you used `mysqli_stmt_store_result` to buffer the entire result set in the statement handle.

If you use `mysqli_stmt_store_result`, `mysqli_stmt_num_rows` may be called immediately.

Parameters

`stmt`	Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

An integer representing the number of rows in result set.

Examples

Example 2.58. Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name LIMIT 20";
if ($stmt = $mysqli->prepare($query)) {
    /* execute query */
    $stmt->execute();
    /* store result */
    $stmt->store_result();
    printf("Number of rows: %d.\n", $stmt->num_rows);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 2.59. Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name LIMIT 20";
if ($stmt = mysqli_prepare($link, $query)) {
    /* execute query */
    mysqli_stmt_execute($stmt);
    /* store result */
    mysqli_stmt_store_result($stmt);
    printf("Number of rows: %d.\n", mysqli_stmt_num_rows($stmt));
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Number of rows: 20.
```

See Also

mysqli_stmt_affected_rows
mysqli_prepare
mysqli_stmt_store_result

2.8.17. mysqli_stmt->param_count, mysqli_stmt_param_count

- mysqli_stmt->param_count

  mysqli_stmt_param_count

  Returns the number of parameter for the given statement

Description

Object oriented style

```
 mysqli_stmt {

  int param_count ;

}
```

Procedural style

```
  int mysqli_stmt_param_count(mysqli_stmt stmt);
```

Returns the number of parameter markers present in the prepared statement.

Parameters

stmt	Procedural style only: A statement identifier returned by mysqli_stmt_init.

Return Values

Returns an integer representing the number of parameters.

Examples

Example 2.60. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if ($stmt = $mysqli->prepare("SELECT Name FROM Country WHERE Name=? OR Code=?")) {
    $marker = $stmt->param_count;
    printf("Statement has %d markers.\n", $marker);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 2.61. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if ($stmt = mysqli_prepare($link, "SELECT Name FROM Country WHERE Name=? OR Code=?")) {
    $marker = mysqli_stmt_param_count($stmt);
    printf("Statement has %d markers.\n", $marker);
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Statement has 2 markers.
```

See Also

mysqli_prepare

2.8.18. mysqli_stmt::prepare, mysqli_stmt_prepare

- `mysqli_stmt::prepare`

  `mysqli_stmt_prepare`

  Prepare an SQL statement for execution

Description

Object oriented style

```
mixed mysqli_stmt::prepare(string query);
```

Procedural style

```
bool mysqli_stmt_prepare(mysqli_stmt stmt,
                         string query);
```

Prepares the SQL query pointed to by the null-terminated string query.

The parameter markers must be bound to application variables using `mysqli_stmt_bind_param` and/or `mysqli_stmt_bind_result` before executing the statement or fetching rows.

> **Note**
>
> In the case where you pass a statement to `mysqli_stmt_prepare` that is longer than `max_allowed_packet` of the server, the returned error codes are different depending on whether you are using MySQL Native Driver (`mysqlnd`) or MySQL Client Library (`libmysql`). The behavior is as follows:
>
> - `mysqlnd` on Linux returns an error code of 1153. The error message means “got a packet bigger than `max_allowed_packet` bytes”.
> - `mysqlnd` on Windows returns an error code 2006. This error message means “server has gone away”.
> - `libmysql` on all platforms returns an error code 2006. This error message means “server has gone away”.

Parameters

`stmt`	Procedural style only: A statement identifier returned by `mysqli_stmt_init`.
`query`	The query, as a string. It must consist of a single SQL statement. You can include one or more parameter markers in the SQL statement by embedding question mark (`?`) characters at the appropriate positions.

> **Note**
>
> You should not add a terminating semicolon or `\g` to the statement.

> **Note**
>
> The markers are legal only in certain places in SQL statements. For example, they are allowed in the VALUES() list of an INSERT statement (to specify column values for a row), or in a comparison with a column in a WHERE clause to specify a comparison value.
>
> However, they are not allowed for identifiers (such as table or column names), in the select list that names the columns to be returned by a SELECT statement), or to specify both operands of a binary operator such as the `=` equal sign. The latter restriction is necessary because it would be impossible to determine the parameter type. In general, parameters are legal only in Data Manipulation Language (DML) statements, and not in Data Definition Language (DDL) statements.

Return Values

Returns TRUE on success or FALSE on failure.

Examples

Example 2.62. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$city = "Amersfoort";
/* create a prepared statement */
$stmt =  $mysqli->stmt_init();
if ($stmt->prepare("SELECT District FROM City WHERE Name=?")) {
    /* bind parameters for markers */
    $stmt->bind_param("s", $city);
    /* execute query */
    $stmt->execute();
    /* bind result variables */
    $stmt->bind_result($district);
    /* fetch value */
    $stmt->fetch();
    printf("%s is in district %s\n", $city, $district);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 2.63. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$city = "Amersfoort";
/* create a prepared statement */
$stmt = mysqli_stmt_init($link);
if (mysqli_stmt_prepare($stmt, 'SELECT District FROM City WHERE Name=?')) {
    /* bind parameters for markers */
    mysqli_stmt_bind_param($stmt, "s", $city);
    /* execute query */
    mysqli_stmt_execute($stmt);
    /* bind result variables */
    mysqli_stmt_bind_result($stmt, $district);
    /* fetch value */
    mysqli_stmt_fetch($stmt);
    printf("%s is in district %s\n", $city, $district);
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Amersfoort is in district Utrecht
```

See Also

mysqli_stmt_init, mysqli_stmt_execute, mysqli_stmt_fetch, mysqli_stmt_bind_param, mysqli_stmt_bind_result mysqli_stmt_close.

2.8.19. mysqli_stmt::reset, mysqli_stmt_reset

- mysqli_stmt::reset

  mysqli_stmt_reset

  Resets a prepared statement

Description

Object oriented style

```
bool mysqli_stmt::reset();
```

Procedural style

```
bool mysqli_stmt_reset(mysqli_stmt stmt);
```

Resets a prepared statement on client and server to state after prepare.

It resets the statement on the server, data sent using mysqli_stmt_send_long_data, unbuffered result sets and current errors. It does not clear bindings or stored result sets. Stored result sets will be cleared when executing the prepared statement (or closing it).

To prepare a statement with another query use function mysqli_stmt_prepare.

Parameters

stmt	Procedural style only: A statement identifier returned by mysqli_stmt_init.

Return Values

Returns TRUE on success or FALSE on failure.

See Also

mysqli_prepare

2.8.20. mysqli_stmt::result_metadata, mysqli_stmt_result_metadata

- mysqli_stmt::result_metadata

  mysqli_stmt_result_metadata

  Returns result set metadata from a prepared statement

Description

Object oriented style

```
mysqli_result mysqli_stmt::result_metadata();
```

Procedural style

```
mysqli_result mysqli_stmt_result_metadata(mysqli_stmt stmt);
```

If a statement passed to `mysqli_prepare` is one that produces a result set, `mysqli_stmt_result_metadata` returns the result object that can be used to process the meta information such as total number of fields and individual field information.

Note

This result set pointer can be passed as an argument to any of the field-based functions that process result set metadata, such as:

- `mysqli_num_fields`
- `mysqli_fetch_field`
- `mysqli_fetch_field_direct`
- `mysqli_fetch_fields`
- `mysqli_field_count`
- `mysqli_field_seek`
- `mysqli_field_tell`
- `mysqli_free_result`

The result set structure should be freed when you are done with it, which you can do by passing it to `mysqli_free_result`

Note

The result set returned by `mysqli_stmt_result_metadata` contains only metadata. It does not contain any row results. The rows are obtained by using the statement handle with `mysqli_stmt_fetch`.

Parameters

`stmt`	Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Returns a result object or `FALSE` if an error occurred.

Examples

Example 2.64. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");
$mysqli->query("DROP TABLE IF EXISTS friends");
$mysqli->query("CREATE TABLE friends (id int, name varchar(20))");
$mysqli->query("INSERT INTO friends VALUES (1,'Hartmut'), (2, 'Ulf')");
$stmt = $mysqli->prepare("SELECT id, name FROM friends");
$stmt->execute();
/* get resultset for metadata */
$result = $stmt->result_metadata();
/* retrieve field information from metadata result set */
$field = $result->fetch_field();
printf("Fieldname: %s\n", $field->name);
/* close resultset */
```

```
$result->close();
/* close connection */
$mysqli->close();
?>
```

Example 2.65. Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "test");
mysqli_query($link, "DROP TABLE IF EXISTS friends");
mysqli_query($link, "CREATE TABLE friends (id int, name varchar(20))");
mysqli_query($link, "INSERT INTO friends VALUES (1,'Hartmut'), (2, 'Ulf')");
$stmt = mysqli_prepare($link, "SELECT id, name FROM friends");
mysqli_stmt_execute($stmt);
/* get resultset for metadata */
$result = mysqli_stmt_result_metadata($stmt);
/* retrieve field information from metadata result set */
$field = mysqli_fetch_field($result);
printf("Fieldname: %s\n", $field->name);
/* close resultset */
mysqli_free_result($result);
/* close connection */
mysqli_close($link);
?>
```

See Also

mysqli_prepare
mysqli_free_result

2.8.21. mysqli_stmt::send_long_data, mysqli_stmt_send_long_data

- mysqli_stmt::send_long_data

  mysqli_stmt_send_long_data

  Send data in blocks

Description

Object oriented style

```
  bool mysqli_stmt::send_long_data(int param_nr,
                                   string data);
```

Procedural style

```
  bool mysqli_stmt_send_long_data(mysqli_stmt stmt,
                                  int param_nr,
                                  string data);
```

Allows to send parameter data to the server in pieces (or chunks), e.g. if the size of a blob exceeds the size of `max_allowed_packet`. This function can be called multiple times to send the parts of a character or binary data value for a column, which must be one of the TEXT or BLOB datatypes.

Parameters

stmt	Procedural style only: A statement identifier returned by `mysqli_stmt_init`.
param_nr	Indicates which parameter to associate the data with. Parameters are numbered beginning with 0.
data	A string containing data to be sent.

Return Values

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

Examples

Example 2.66. Object oriented style

```
<?php
$stmt = $mysqli->prepare("INSERT INTO messages (message) VALUES (?)");
$null = NULL;
$stmt->bind_param("b", $null);
$fp = fopen("messages.txt", "r");
while (!feof($fp)) {
    $stmt->send_long_data(0, fread($fp, 8192));
}
fclose($fp);
$stmt->execute();
?>
```

See Also

`mysqli_prepare`
`mysqli_stmt_bind_param`

2.8.22. `mysqli_stmt::sqlstate`, `mysqli_stmt_sqlstate`

- `mysqli_stmt::sqlstate`

  `mysqli_stmt_sqlstate`

  Returns SQLSTATE error from previous statement operation

Description

Object oriented style

```
 mysqli_stmt {

  string sqlstate ;

}
```

Procedural style

```
  string mysqli_stmt_sqlstate(mysqli_stmt stmt);
```

Returns a string containing the SQLSTATE error code for the most recently invoked prepared statement function that can succeed or fail. The error code consists of five characters. `'00000'` means no error. The values are specified by ANSI SQL and ODBC. For a list of possible values, see http://dev.mysql.com/doc/mysql/en/error-handling.html.

Parameters

stmt	Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Returns a string containing the SQLSTATE error code for the last error. The error code consists of five characters. `'00000'` means no error.

Notes

Note

Note that not all MySQL errors are yet mapped to SQLSTATE's. The value `HY000` (general error) is used for unmapped errors.

Examples

Example 2.67. Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TABLE myCountry LIKE Country");
$mysqli->query("INSERT INTO myCountry SELECT * FROM Country");
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = $mysqli->prepare($query)) {
    /* drop table */
    $mysqli->query("DROP TABLE myCountry");
    /* execute query */
    $stmt->execute();
    printf("Error: %s.\n", $stmt->sqlstate);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TABLE myCountry LIKE Country");
mysqli_query($link, "INSERT INTO myCountry SELECT * FROM Country");
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = mysqli_prepare($link, $query)) {
    /* drop table */
    mysqli_query($link, "DROP TABLE myCountry");
    /* execute query */
    mysqli_stmt_execute($stmt);
    printf("Error: %s.\n", mysqli_stmt_sqlstate($stmt));
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Error: 42S02.
```

See Also

mysqli_stmt_errno
mysqli_stmt_error

2.8.23. mysqli_stmt::store_result, mysqli_stmt_store_result

- mysqli_stmt::store_result

  mysqli_stmt_store_result

  Transfers a result set from a prepared statement

Description

Object oriented style

```
bool mysqli_stmt::store_result();
```

Procedural style

```
bool mysqli_stmt_store_result(mysqli_stmt stmt);
```

You must call `mysqli_stmt_store_result` for every query that successfully produces a result set (`SELECT, SHOW, DESCRIBE, EXPLAIN`), and only if you want to buffer the complete result set by the client, so that the subsequent `mysqli_stmt_fetch` call returns buffered data.

> **Note**
>
> It is unnecessary to call `mysqli_stmt_store_result` for other queries, but if you do, it will not harm or cause any notable performance in all cases. You can detect whether the query produced a result set by checking if `mysqli_stmt_result_metadata` returns NULL.

Parameters

stmt	Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

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

Examples

Example 2.68. Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
```

```
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name LIMIT 20";
if ($stmt = $mysqli->prepare($query)) {
    /* execute query */
    $stmt->execute();
    /* store result */
    $stmt->store_result();
    printf("Number of rows: %d.\n", $stmt->num_rows);
    /* free result */
    $stmt->free_result();
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name LIMIT 20";
if ($stmt = mysqli_prepare($link, $query)) {
    /* execute query */
    mysqli_stmt_execute($stmt);
    /* store result */
    mysqli_stmt_store_result($stmt);
    printf("Number of rows: %d.\n", mysqli_stmt_num_rows($stmt));
    /* free result */
    mysqli_stmt_free_result($stmt);
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Number of rows: 20.
```

See Also

`mysqli_prepare`
`mysqli_stmt_result_metadata`
`mysqli_stmt_fetch`

2.9. The MySQLi_Result class (`MySQLi_Result`)

Represents the result set obtained from a query against the database.

```
 MySQLi_Result {
 MySQLi_Result
      Properties
```

```
  int current_field ;

  int field_count ;

  array lengths ;

  int num_rows ;
Methods
  int mysqli_field_tell(mysqli_result result);

  bool mysqli_result::data_seek(int offset);

  mixed mysqli_result::fetch_all(int resulttype= =MYSQLI_NUM);

  mixed mysqli_result::fetch_array(int resulttype= =MYSQLI_BOTH);

  array mysqli_result::fetch_assoc();

  object mysqli_result::fetch_field_direct(int fieldnr);

  object mysqli_result::fetch_field();

  array mysqli_result::fetch_fields();

  object mysqli_result::fetch_object(string class_name,
                                     array params);

  mixed mysqli_result::fetch_row();

  int mysqli_num_fields(mysqli_result result);

  bool mysqli_result::field_seek(int fieldnr);

  void mysqli_result::free();

  array mysqli_fetch_lengths(mysqli_result result);

  int mysqli_num_rows(mysqli_result result);
}
```

2.9.1. mysqli_result->current_field, mysqli_field_tell

- mysqli_result->current_field

  mysqli_field_tell

  Get current field offset of a result pointer

Description

Object oriented style

```
 mysqli_result {

  int current_field ;

}
```

Procedural style

```
  int mysqli_field_tell(mysqli_result result);
```

Returns the position of the field cursor used for the last `mysqli_fetch_field` call. This value can be used as an argument to `mysqli_field_seek`.

Parameters

`result`	Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

Returns current offset of field cursor.

Examples

Example 2.69. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";
if ($result = $mysqli->query($query)) {
    /* Get field information for all columns */
    while ($finfo = $result->fetch_field()) {
        /* get fieldpointer offset */
        $currentfield = $result->current_field;
        printf("Column %d:\n", $currentfield);
        printf("Name:     %s\n", $finfo->name);
        printf("Table:    %s\n", $finfo->table);
        printf("max. Len: %d\n", $finfo->max_length);
        printf("Flags:    %d\n", $finfo->flags);
        printf("Type:     %d\n\n", $finfo->type);
    }
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";
if ($result = mysqli_query($link, $query)) {
    /* Get field information for all fields */
    while ($finfo = mysqli_fetch_field($result)) {
        /* get fieldpointer offset */
        $currentfield = mysqli_field_tell($result);
        printf("Column %d:\n", $currentfield);
        printf("Name:     %s\n", $finfo->name);
        printf("Table:    %s\n", $finfo->table);
        printf("max. Len: %d\n", $finfo->max_length);
        printf("Flags:    %d\n", $finfo->flags);
        printf("Type:     %d\n\n", $finfo->type);
    }
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Column 1:
Name:     Name
Table:    Country
max. Len: 11
Flags:    1
Type:     254
Column 2:
Name:     SurfaceArea
Table:    Country
max. Len: 10
Flags:    32769
Type:     4
```

See Also

mysqli_fetch_field
mysqli_field_seek

2.9.2. mysqli_result::data_seek, mysqli_data_seek

- mysqli_result::data_seek

  mysqli_data_seek

  Adjusts the result pointer to an arbitary row in the result

Description

Object oriented style

```
bool mysqli_result::data_seek(int offset);
```

Procedural style

```
bool mysqli_data_seek(mysqli_result result,
                      int offset);
```

The mysqli_data_seek function seeks to an arbitrary result pointer specified by the *offset* in the result set.

Parameters

result	Procedural style only: A result set identifier returned by mysqli_query, mysqli_store_result or mysqli_use_result.

offset	The field offset. Must be between zero and the total number of rows minus one (0..mysqli_num_rows - 1).

Return Values

Returns TRUE on success or FALSE on failure.

Notes

> **Note**
>
> This function can only be used with buffered results attained from the use of the mysqli_store_result or mysqli_query functions.

Examples

Example 2.70. Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name";
if ($result = $mysqli->query( $query)) {
    /* seek to row no. 400 */
    $result->data_seek(399);
    /* fetch row */
    $row = $result->fetch_row();
    printf ("City: %s  Countrycode: %s\n", $row[0], $row[1]);
    /* free result set*/
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name";
if ($result = mysqli_query($link, $query)) {
    /* seek to row no. 400 */
    mysqli_data_seek($result, 399);
    /* fetch row */
    $row = mysqli_fetch_row($result);
    printf ("City: %s  Countrycode: %s\n", $row[0], $row[1]);
    /* free result set*/
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
City: Benin City  Countrycode: NGA
```

See Also

mysqli_store_result
mysqli_fetch_row
mysqli_fetch_array
mysqli_fetch_assoc
mysqli_fetch_object
mysqli_query
mysqli_num_rows

2.9.3. mysqli_result::fetch_all, mysqli_fetch_all

- mysqli_result::fetch_all

  mysqli_fetch_all

  Fetches all result rows as an associative array, a numeric array, or both

Description

Object oriented style

```
mixed mysqli_result::fetch_all(int resulttype= =MYSQLI_NUM);
```

Procedural style

```
mixed mysqli_fetch_all(mysqli_result result,
                       int resulttype= =MYSQLI_NUM);
```

mysqli_fetch_all fetches all result rows and returns the result set as an associative array, a numeric array, or both.

Parameters

result	Procedural style only: A result set identifier returned by mysqli_query, mysqli_store_result or mysqli_use_result.
resulttype	This optional parameter is a constant indicating what type of array should be produced from the current row data. The possible values for this parameter are the constants MYSQLI_ASSOC , MYSQLI_NUM , or MYSQLI_BOTH .

Return Values

Returns an array of associative or numeric arrays holding result rows.

MySQL Native Driver Only

Available only with mysqlnd.

As mysqli_fetch_all returns all the rows as an array in a single step, it may consume more memory than some similar functions such as mysqli_fetch_array, which only returns one row at a time from the result set. Further, if you need to iterate over the result set, you will need a looping construct that will further impact performance. For these reasons mysqli_fetch_all should only be used in those situations where the fetched result set will be sent to another layer for processing.

See Also

mysqli_fetch_array
mysqli_query

2.9.4. mysqli_result::fetch_array, mysqli_fetch_array

- mysqli_result::fetch_array

  mysqli_fetch_array

  Fetch a result row as an associative, a numeric array, or both

Description

Object oriented style

```
mixed mysqli_result::fetch_array(int resulttype= =MYSQLI_BOTH);
```

Procedural style

```
mixed mysqli_fetch_array(mysqli_result result,
                         int resulttype= =MYSQLI_BOTH);
```

Returns an array that corresponds to the fetched row or `NULL` if there are no more rows for the resultset represented by the *result* parameter.

`mysqli_fetch_array` is an extended version of the `mysqli_fetch_row` function. In addition to storing the data in the numeric indices of the result array, the `mysqli_fetch_array` function can also store the data in associative indices, using the field names of the result set as keys.

> **Note**
>
> Field names returned by this function are *case-sensitive*.

> **Note**
>
> This function sets NULL fields to the PHP `NULL` value.

If two or more columns of the result have the same field names, the last column will take precedence and overwrite the earlier data. In order to access multiple columns with the same name, the numerically indexed version of the row must be used.

Parameters

result	Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.
resulttype	This optional parameter is a constant indicating what type of array should be produced from the current row data. The possible values for this parameter are the constants `MYSQLI_ASSOC` , `MYSQLI_NUM` , or `MYSQLI_BOTH` . By using the `MYSQLI_ASSOC` constant this function will behave identically to the `mysqli_fetch_assoc`, while `MYSQLI_NUM` will behave identically to the `mysqli_fetch_row` function. The final option `MYSQLI_BOTH` will create a single array with the attributes of both.

Return Values

Returns an array of strings that corresponds to the fetched row or `NULL` if there are no more rows in resultset.

Examples

Example 2.71. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID LIMIT 3";
$result = $mysqli->query($query);
/* numeric array */
$row = $result->fetch_array(MYSQLI_NUM);
printf ("%s (%s)\n", $row[0], $row[1]);
/* associative array */
$row = $result->fetch_array(MYSQLI_ASSOC);
printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);
/* associative and numeric array */
$row = $result->fetch_array(MYSQLI_BOTH);
printf ("%s (%s)\n", $row[0], $row["CountryCode"]);
/* free result set */
$result->close();
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID LIMIT 3";
$result = mysqli_query($link, $query);
/* numeric array */
$row = mysqli_fetch_array($result, MYSQLI_NUM);
printf ("%s (%s)\n", $row[0], $row[1]);
/* associative array */
$row = mysqli_fetch_array($result, MYSQLI_ASSOC);
printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);
/* associative and numeric array */
$row = mysqli_fetch_array($result, MYSQLI_BOTH);
printf ("%s (%s)\n", $row[0], $row["CountryCode"]);
/* free result set */
mysqli_free_result($result);
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Kabul (AFG)
Qandahar (AFG)
Herat (AFG)
```

See Also

`mysqli_fetch_assoc`
`mysqli_fetch_row`
`mysqli_fetch_object`
`mysqli_query`

mysqli_data_seek

2.9.5. mysqli_result::fetch_assoc, mysqli_fetch_assoc

- mysqli_result::fetch_assoc

  mysqli_fetch_assoc

  Fetch a result row as an associative array

Description

Object oriented style

```
array mysqli_result::fetch_assoc();
```

Procedural style

```
array mysqli_fetch_assoc(mysqli_result result);
```

Returns an associative array that corresponds to the fetched row or NULL if there are no more rows.

> **Note**
>
> Field names returned by this function are *case-sensitive*.

> **Note**
>
> This function sets NULL fields to the PHP NULL value.

Parameters

result	Procedural style only: A result set identifier returned by mysqli_query, mysqli_store_result or mysqli_use_result.

Return Values

Returns an associative array of strings representing the fetched row in the result set, where each key in the array represents the name of one of the result set's columns or NULL if there are no more rows in resultset.

If two or more columns of the result have the same field names, the last column will take precedence. To access the other column(s) of the same name, you either need to access the result with numeric indices by using mysqli_fetch_row or add alias names.

Examples

Example 2.72. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
if ($result = $mysqli->query($query)) {
    /* fetch associative array */
    while ($row = $result->fetch_assoc()) {
        printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);
    }
    /* free result set */
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
if ($result = mysqli_query($link, $query)) {
    /* fetch associative array */
    while ($row = mysqli_fetch_assoc($result)) {
        printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);
    }
    /* free result set */
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Pueblo (USA)
Arvada (USA)
Cape Coral (USA)
Green Bay (USA)
Santa Clara (USA)
```

See Also

mysqli_fetch_array
mysqli_fetch_row
mysqli_fetch_object
mysqli_query
mysqli_data_seek

2.9.6. mysqli_result::fetch_field_direct, mysqli_fetch_field_direct

- mysqli_result::fetch_field_direct

  mysqli_fetch_field_direct

  Fetch meta-data for a single field

Description

Object oriented style

```
object mysqli_result::fetch_field_direct(int fieldnr);
```

Procedural style

```
object mysqli_fetch_field_direct(mysqli_result result,
                                 int fieldnr);
```

Returns an object which contains field definition information from the specified result set.

Parameters

result	Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.
fieldnr	The field number. This value must be in the range from `0` to `number of fields - 1`.

Return Values

Returns an object which contains field definition information or `FALSE` if no field information for specified `fieldnr` is available.

Table 2.8. Object attributes

Attribute	Description
name	The name of the column
orgname	Original column name if an alias was specified
table	The name of the table this field belongs to (if not calculated)
orgtable	Original table name if an alias was specified
def	The default value for this field, represented as a string
max_length	The maximum width of the field for the result set.
length	The width of the field, as specified in the table definition.
charsetnr	The character set number for the field.
flags	An integer representing the bit-flags for the field.
type	The data type used for this field
decimals	The number of decimals used (for integer fields)

Examples

Example 2.73. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Name LIMIT 5";
if ($result = $mysqli->query($query)) {
    /* Get field information for column 'SurfaceArea' */
    $finfo = $result->fetch_field_direct(1);
    printf("Name:     %s\n", $finfo->name);
    printf("Table:    %s\n", $finfo->table);
    printf("max. Len: %d\n", $finfo->max_length);
    printf("Flags:    %d\n", $finfo->flags);
    printf("Type:     %d\n", $finfo->type);
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Name LIMIT 5";
if ($result = mysqli_query($link, $query)) {
    /* Get field information for column 'SurfaceArea' */
    $finfo = mysqli_fetch_field_direct($result, 1);
    printf("Name:     %s\n", $finfo->name);
    printf("Table:    %s\n", $finfo->table);
    printf("max. Len: %d\n", $finfo->max_length);
    printf("Flags:    %d\n", $finfo->flags);
    printf("Type:     %d\n", $finfo->type);
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Name:     SurfaceArea
Table:    Country
max. Len: 10
Flags:    32769
Type:     4
```

See Also

`mysqli_num_fields`
`mysqli_fetch_field`
`mysqli_fetch_fields`

2.9.7. `mysqli_result::fetch_field`, `mysqli_fetch_field`

- `mysqli_result::fetch_field`

  `mysqli_fetch_field`

  Returns the next field in the result set

Description

Object oriented style

```
object mysqli_result::fetch_field();
```

Procedural style

```
object mysqli_fetch_field(mysqli_result result);
```

Returns the definition of one column of a result set as an object. Call this function repeatedly to retrieve information about all columns in the result set.

Parameters

result	Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

Returns an object which contains field definition information or `FALSE` if no field information is available.

Table 2.9. Object properties

Property	**Description**
name	The name of the column
orgname	Original column name if an alias was specified
table	The name of the table this field belongs to (if not calculated)
orgtable	Original table name if an alias was specified
max_length	The maximum width of the field for the result set.
length	The width of the field, as specified in the table definition.
charsetnr	The character set number for the field.
flags	An integer representing the bit-flags for the field.
type	The data type used for this field
decimals	The number of decimals used (for integer fields)

Examples

Example 2.74. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";
if ($result = $mysqli->query($query)) {
    /* Get field information for all columns */
    while ($finfo = $result->fetch_field()) {
        printf("Name:     %s\n", $finfo->name);
        printf("Table:    %s\n", $finfo->table);
        printf("max. Len: %d\n", $finfo->max_length);
        printf("Flags:    %d\n", $finfo->flags);
        printf("Type:     %d\n\n", $finfo->type);
    }
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";
if ($result = mysqli_query($link, $query)) {
    /* Get field information for all fields */
    while ($finfo = mysqli_fetch_field($result)) {
        printf("Name:     %s\n", $finfo->name);
        printf("Table:    %s\n", $finfo->table);
        printf("max. Len: %d\n", $finfo->max_length);
        printf("Flags:    %d\n", $finfo->flags);
        printf("Type:     %d\n\n", $finfo->type);
    }
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Name:     Name
Table:    Country
max. Len: 11
Flags:    1
Type:     254
Name:     SurfaceArea
Table:    Country
max. Len: 10
Flags:    32769
Type:     4
```

See Also

mysqli_num_fields
mysqli_fetch_field_direct
mysqli_fetch_fields
mysqli_field_seek

2.9.8. mysqli_result::fetch_fields, mysqli_fetch_fields

- mysqli_result::fetch_fields

  mysqli_fetch_fields

  Returns an array of objects representing the fields in a result set

Description

Object oriented style

```
array mysqli_result::fetch_fields();
```

Procedural style

```
array mysqli_fetch_fields(mysqli_result result);
```

This function serves an identical purpose to the mysqli_fetch_field function with the single difference that, instead of returning one object at a time for each field, the columns are returned as an array of objects.

Parameters

result	Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

Returns an array of objects which contains field definition information or `FALSE` if no field information is available.

Table 2.10. Object properties

Property	**Description**
name	The name of the column
orgname	Original column name if an alias was specified
table	The name of the table this field belongs to (if not calculated)
orgtable	Original table name if an alias was specified
max_length	The maximum width of the field for the result set.
length	The width of the field, as specified in the table definition.
charsetnr	The character set number for the field.
flags	An integer representing the bit-flags for the field.
type	The data type used for this field
decimals	The number of decimals used (for integer fields)

Examples

Example 2.75. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";
if ($result = $mysqli->query($query)) {
    /* Get field information for all columns */
    $finfo = $result->fetch_fields();
    foreach ($finfo as $val) {
        printf("Name:     %s\n", $val->name);
        printf("Table:    %s\n", $val->table);
        printf("max. Len: %d\n", $val->max_length);
        printf("Flags:    %d\n", $val->flags);
        printf("Type:     %d\n\n", $val->type);
    }
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";
if ($result = mysqli_query($link, $query)) {
    /* Get field information for all columns */
    $finfo = mysqli_fetch_fields($result);
    foreach ($finfo as $val) {
        printf("Name:     %s\n", $val->name);
        printf("Table:    %s\n", $val->table);
        printf("max. Len: %d\n", $val->max_length);
        printf("Flags:    %d\n", $val->flags);
        printf("Type:     %d\n\n", $val->type);
    }
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Name:     Name
Table:    Country
max. Len: 11
Flags:    1
Type:     254
Name:     SurfaceArea
Table:    Country
max. Len: 10
Flags:    32769
Type:     4
```

See Also

`mysqli_num_fields`
`mysqli_fetch_field_direct`
`mysqli_fetch_field`

2.9.9. mysqli_result::fetch_object, mysqli_fetch_object

- `mysqli_result::fetch_object`

  `mysqli_fetch_object`

  Returns the current row of a result set as an object

Description

Object oriented style

```
object mysqli_result::fetch_object(string class_name,
                                   array params);
```

Procedural style

```
object mysqli_fetch_object(mysqli_result result,
                           string class_name,
                           array params);
```

The `mysqli_fetch_object` will return the current row result set as an object where the attributes of the object represent the names of the fields found within the result set.

Parameters

result	Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.
class_name	The name of the class to instantiate, set the properties of and return. If not specified, a `stdClass` object is returned.
params	An optional array of parameters to pass to the constructor for *class_name* objects.

Return Values

Returns an object with string properties that corresponds to the fetched row or `NULL` if there are no more rows in resultset.

Note

Field names returned by this function are *case-sensitive*.

Note

This function sets NULL fields to the PHP `NULL` value.

Changelog

Version	Description
5.0.0	Added the ability to return as a different object.

Examples

Example 2.76. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
if ($result = $mysqli->query($query)) {
    /* fetch object array */
    while ($obj = $result->fetch_object()) {
        printf ("%s (%s)\n", $obj->Name, $obj->CountryCode);
    }
    /* free result set */
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
if ($result = mysqli_query($link, $query)) {
    /* fetch associative array */
    while ($obj = mysqli_fetch_object($result)) {
        printf ("%s (%s)\n", $obj->Name, $obj->CountryCode);
    }
    /* free result set */
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Pueblo (USA)
Arvada (USA)
Cape Coral (USA)
Green Bay (USA)
Santa Clara (USA)
```

See Also

`mysqli_fetch_array`
`mysqli_fetch_assoc`
`mysqli_fetch_row`
`mysqli_query`
`mysqli_data_seek`

2.9.10. `mysqli_result::fetch_row`, `mysqli_fetch_row`

- `mysqli_result::fetch_row`

  `mysqli_fetch_row`

  Get a result row as an enumerated array

Description

Object oriented style

```
mixed mysqli_result::fetch_row();
```

Procedural style

```
mixed mysqli_fetch_row(mysqli_result result);
```

Fetches one row of data from the result set and returns it as an enumerated array, where each column is stored in an array offset starting from 0 (zero). Each subsequent call to this function will return the next row within the result set, or `NULL` if there are no more rows.

Parameters

`result`	Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

mysqli_fetch_row returns an array of strings that corresponds to the fetched row or NULL if there are no more rows in result set.

Note

This function sets NULL fields to the PHP NULL value.

Examples

Example 2.77. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
if ($result = $mysqli->query($query)) {
    /* fetch object array */
    while ($row = $result->fetch_row()) {
        printf ("%s (%s)\n", $row[0], $row[1]);
    }
    /* free result set */
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
if ($result = mysqli_query($link, $query)) {
    /* fetch associative array */
    while ($row = mysqli_fetch_row($result)) {
        printf ("%s (%s)\n", $row[0], $row[1]);
    }
    /* free result set */
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Pueblo (USA)
Arvada (USA)
Cape Coral (USA)
Green Bay (USA)
Santa Clara (USA)
```

See Also

mysqli_fetch_array
mysqli_fetch_assoc

mysqli_fetch_object
mysqli_query
mysqli_data_seek

2.9.11. mysqli_result->field_count, mysqli_num_fields

- mysqli_result->field_count

  mysqli_num_fields

  Get the number of fields in a result

Description

Object oriented style

```
 mysqli_result {

  int field_count ;

}
```

Procedural style

```
  int mysqli_num_fields(mysqli_result result);
```

Returns the number of fields from specified result set.

Parameters

result	Procedural style only: A result set identifier returned by mysqli_query, mysqli_store_result or mysqli_use_result.

Return Values

The number of fields from a result set.

Examples

Example 2.78. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if ($result = $mysqli->query("SELECT * FROM City ORDER BY ID LIMIT 1")) {
    /* determine number of fields in result set */
    $field_cnt = $result->field_count;
    printf("Result set has %d fields.\n", $field_cnt);
    /* close result set */
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if ($result = mysqli_query($link, "SELECT * FROM City ORDER BY ID LIMIT 1")) {
    /* determine number of fields in result set */
    $field_cnt = mysqli_num_fields($result);
    printf("Result set has %d fields.\n", $field_cnt);
    /* close result set */
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Result set has 5 fields.
```

See Also

`mysqli_fetch_field`

2.9.12. `mysqli_result::field_seek`, `mysqli_field_seek`

- `mysqli_result::field_seek`

  `mysqli_field_seek`

  Set result pointer to a specified field offset

Description

Object oriented style

```
bool mysqli_result::field_seek(int fieldnr);
```

Procedural style

```
bool mysqli_field_seek(mysqli_result result,
                       int fieldnr);
```

Sets the field cursor to the given offset. The next call to `mysqli_fetch_field` will retrieve the field definition of the column associated with that offset.

> **Note**
>
> To seek to the beginning of a row, pass an offset value of zero.

Parameters

`result`	Procedural style only: A result set identifier returned by `mysqli_query`,

	mysqli_store_result or mysqli_use_result.
fieldnr	The field number. This value must be in the range from 0 to number of fields - 1.

Return Values

Returns TRUE on success or FALSE on failure.

Examples

Example 2.79. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";
if ($result = $mysqli->query($query)) {
    /* Get field information for 2nd column */
    $result->field_seek(1);
    $finfo = $result->fetch_field();
    printf("Name:     %s\n", $finfo->name);
    printf("Table:    %s\n", $finfo->table);
    printf("max. Len: %d\n", $finfo->max_length);
    printf("Flags:    %d\n", $finfo->flags);
    printf("Type:     %d\n\n", $finfo->type);
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";
if ($result = mysqli_query($link, $query)) {
    /* Get field information for 2nd column */
    mysqli_field_seek($result, 1);
    $finfo = mysqli_fetch_field($result);
    printf("Name:     %s\n", $finfo->name);
    printf("Table:    %s\n", $finfo->table);
    printf("max. Len: %d\n", $finfo->max_length);
    printf("Flags:    %d\n", $finfo->flags);
    printf("Type:     %d\n\n", $finfo->type);
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Name:     SurfaceArea
Table:    Country
max. Len: 10
Flags:    32769
Type:     4
```

See Also

mysqli_fetch_field

2.9.13. mysqli_result::free, mysqli_free_result

- mysqli_result::free

  mysqli_free_result

  Frees the memory associated with a result

Description

Object oriented style

```
void mysqli_result::free();
```

```
void mysqli_result::close();
```

```
void mysqli_result::free_result();
```

Procedural style

```
void mysqli_free_result(mysqli_result result);
```

Frees the memory associated with the result.

> **Note**
>
> You should always free your result with mysqli_free_result, when your result object is not needed anymore.

Parameters

result	Procedural style only: A result set identifier returned by mysqli_query, mysqli_store_result or mysqli_use_result.

Return Values

No value is returned.

See Also

mysqli_query
mysqli_stmt_store_result
mysqli_store_result
mysqli_use_result

2.9.14. mysqli_result->lengths, mysqli_fetch_lengths

- mysqli_result->lengths

mysqli_fetch_lengths

Returns the lengths of the columns of the current row in the result set

Description

Object oriented style

```
 mysqli_result {

  array lengths ;

}
```

Procedural style

```
  array mysqli_fetch_lengths(mysqli_result result);
```

The `mysqli_fetch_lengths` function returns an array containing the lengths of every column of the current row within the result set.

Parameters

result	Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

An array of integers representing the size of each column (not including any terminating null characters). `FALSE` if an error occurred.

`mysqli_fetch_lengths` is valid only for the current row of the result set. It returns `FALSE` if you call it before calling mysqli_fetch_row/array/object or after retrieving all rows in the result.

Examples

Example 2.80. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT * from Country ORDER BY Code LIMIT 1";
if ($result = $mysqli->query($query)) {
    $row = $result->fetch_row();
    /* display column lengths */
    foreach ($result->lengths as $i => $val) {
        printf("Field %2d has Length %2d\n", $i+1, $val);
    }
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT * from Country ORDER BY Code LIMIT 1";
if ($result = mysqli_query($link, $query)) {
    $row = mysqli_fetch_row($result);
    /* display column lengths */
    foreach (mysqli_fetch_lengths($result) as $i => $val) {
        printf("Field %2d has Length %2d\n", $i+1, $val);
    }
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Field  1 has Length  3
Field  2 has Length  5
Field  3 has Length 13
Field  4 has Length  9
Field  5 has Length  6
Field  6 has Length  1
Field  7 has Length  6
Field  8 has Length  4
Field  9 has Length  6
Field 10 has Length  6
Field 11 has Length  5
Field 12 has Length 44
Field 13 has Length  7
Field 14 has Length  3
Field 15 has Length  2
```

2.9.15. mysqli_result->num_rows, mysqli_num_rows

- mysqli_result->num_rows

  mysqli_num_rows

  Gets the number of rows in a result

Description

Object oriented style

```
 mysqli_result {

  int num_rows ;

}
```

Procedural style

```
  int mysqli_num_rows(mysqli_result result);
```

Returns the number of rows in the result set.

The use of mysqli_num_rows depends on whether you use buffered or unbuffered result sets. In case you use unbuffered resultsets mysqli_num_rows will not return the correct number of rows until all the rows in the result have been retrieved.

Parameters

result	Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

Returns number of rows in the result set.

Note

If the number of rows is greater than maximal int value, the number will be returned as a string.

Examples

Example 2.81. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if ($result = $mysqli->query("SELECT Code, Name FROM Country ORDER BY Name")) {
    /* determine number of rows result set */
    $row_cnt = $result->num_rows;
    printf("Result set has %d rows.\n", $row_cnt);
    /* close result set */
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if ($result = mysqli_query($link, "SELECT Code, Name FROM Country ORDER BY Name")) {
    /* determine number of rows result set */
    $row_cnt = mysqli_num_rows($result);
    printf("Result set has %d rows.\n", $row_cnt);
    /* close result set */
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Result set has 239 rows.
```

See Also

`mysqli_affected_rows`

mysqli_store_result
mysqli_use_result
mysqli_query

2.10. The MySQLi_Driver class (MySQLi_Driver)

MySQLi Driver.

```
 MySQLi_Driver {
 MySQLi_Driver
      Properties

  public readonly string client_info ;


  public readonly string client_version ;


  public readonly string driver_version ;


  public readonly string embedded ;


  public bool reconnect ;


  public int report_mode ;

Methods

  void mysqli_driver::embedded_server_end();


  bool mysqli_driver::embedded_server_start(bool start,
                                            array arguments,
                                            array groups);

}
```

client_info	The Client API header version
client_version	The Client version
driver_version	The MySQLi Driver version
embedded	Wether MySQLi Embedded support is enabled
reconnect	Allow or prevent reconnect (see the mysqli.reconnect INI directive)
report_mode	Set to MYSQLI_REPORT_OFF , MYSQLI_REPORT_ALL or any combination of MYSQLI_REPORT_STRICT (throw Exceptions for errors), MYSQLI_REPORT_ERROR (report errors) and MYSQLI_REPORT_INDEX (errors regarding indexes). See also mysqli_report.

2.10.1. mysqli_driver::embedded_server_end, mysqli_embedded_server_end

- mysqli_driver::embedded_server_end

  mysqli_embedded_server_end

  Stop embedded server

Description

Object oriented style

```
void mysqli_driver::embedded_server_end();
```

Procedural style

```
void mysqli_embedded_server_end();
```

> **Warning**
>
> This function is currently not documented; only its argument list is available.

2.10.2. mysqli_driver::embedded_server_start, mysqli_embedded_server_start

- mysqli_driver::embedded_server_start

  mysqli_embedded_server_start

  Initialize and start embedded server

Description

Object oriented style

```
bool mysqli_driver::embedded_server_start(bool start,
                                          array arguments,
                                          array groups);
```

Procedural style

```
bool mysqli_embedded_server_start(bool start,
                                  array arguments,
                                  array groups);
```

> **Warning**
>
> This function is currently not documented; only its argument list is available.

2.11. The MySQLi_Warning class (MySQLi_Warning)

Represents a MySQL warning.

```
mysqli_warning {
mysqli_warning
     Properties

  public message ;


  public sqlstate ;


  public errno ;

Methods

  __construct();
  public void next();
}
```

`message`	Message string
`sqlstate`	SQL state
`errno`	Error number

2.11.1. mysqli_warning::__construct

- `mysqli_warning::__construct`

  The __construct purpose

Description

```
mysqli_warning::__construct();
```

Warning

This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

2.11.2. mysqli_warning::next

- `mysqli_warning::next`

  The next purpose

Description

```
public void mysqli_warning::next();
```

Warning

This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

2.12. Aliases and deprecated Mysqli Functions

2.12.1. `mysqli_bind_param`

- `mysqli_bind_param`

  Alias for `mysqli_stmt_bind_param`

Description

This function is an alias of `mysqli_stmt_bind_param`.

Notes

> **Note**
>
> `mysqli_bind_param` is deprecated and will be removed.

See Also

`mysqli_stmt_bind_param`

2.12.2. `mysqli_bind_result`

- `mysqli_bind_result`

  Alias for `mysqli_stmt_bind_result`

Description

This function is an alias of `mysqli_stmt_bind_result`.

Notes

> **Note**
>
> `mysqli_bind_result` is deprecated and will be removed.

See Also

`mysqli_stmt_bind_result`

2.12.3. `mysqli_client_encoding`

- `mysqli_client_encoding`

  Alias of `mysqli_character_set_name`

Description

This function is an alias of `mysqli_character_set_name`.

See Also

`mysqli_real_escape_string`

2.12.4. mysqli_connect

- `mysqli_connect`

  Alias of `mysqli::__construct`

Description

This function is an alias of: `mysqli::__construct`

2.12.5. mysqli_disable_reads_from_master, mysqli->disable_reads_from_master

- `mysqli_disable_reads_from_master`

  `mysqli->disable_reads_from_master`

  Disable reads from master

Description

Object oriented style

```
void mysqli::disable_reads_from_master();
```

Procedural style

```
bool mysqli_disable_reads_from_master(mysqli link);
```

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

2.12.6. mysqli_disable_rpl_parse

- `mysqli_disable_rpl_parse`

  Disable RPL parse

Description

```
bool mysqli_disable_rpl_parse(mysqli link);
```

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

2.12.7. mysqli_enable_reads_from_master

- `mysqli_enable_reads_from_master`

  Enable reads from master

Description

```
bool mysqli_enable_reads_from_master(mysqli link);
```

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

2.12.8. mysqli_enable_rpl_parse

- `mysqli_enable_rpl_parse`

  Enable RPL parse

Description

```
bool mysqli_enable_rpl_parse(mysqli link);
```

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

2.12.9. mysqli_escape_string

- `mysqli_escape_string`

  Alias of `mysqli_real_escape_string`

Description

This function is an alias of: mysqli_real_escape_string.

2.12.10. mysqli_execute

- mysqli_execute

  Alias for mysqli_stmt_execute

Description

This function is an alias of mysqli_stmt_execute.

Notes

> **Note**
>
> mysqli_execute is deprecated and will be removed.

See Also

mysqli_stmt_execute

2.12.11. mysqli_fetch

- mysqli_fetch

  Alias for mysqli_stmt_fetch

Description

This function is an alias of mysqli_stmt_fetch.

Notes

> **Note**
>
> mysqli_fetch is deprecated and will be removed.

See Also

mysqli_stmt_fetch

2.12.12. mysqli_get_cache_stats

- mysqli_get_cache_stats

  Returns client Zval cache statistics

Description

```
array mysqli_get_cache_stats();
```

Warning

This function is currently not documented; only its argument list is available.

Returns client Zval cache statistics. Available only with mysqlnd.

Parameters

Return Values

Returns an array with client Zval cache stats if success, `FALSE` otherwise.

Examples

Example 2.82. A `mysqli_get_cache_stats` example

```
<?php
$link = mysqli_connect();
print_r(mysqli_get_cache_stats());
?>
```

The above example will output something similar to:

```
Array
(
    [bytes_sent] => 43
    [bytes_received] => 80
    [packets_sent] => 1
    [packets_received] => 2
    [protocol_overhead_in] => 8
    [protocol_overhead_out] => 4
    [bytes_received_ok_packet] => 11
    [bytes_received_eof_packet] => 0
    [bytes_received_rset_header_packet] => 0
    [bytes_received_rset_field_meta_packet] => 0
    [bytes_received_rset_row_packet] => 0
    [bytes_received_prepare_response_packet] => 0
    [bytes_received_change_user_packet] => 0
    [packets_sent_command] => 0
    [packets_received_ok] => 1
    [packets_received_eof] => 0
    [packets_received_rset_header] => 0
    [packets_received_rset_field_meta] => 0
    [packets_received_rset_row] => 0
    [packets_received_prepare_response] => 0
    [packets_received_change_user] => 0
    [result_set_queries] => 0
    [non_result_set_queries] => 0
    [no_index_used] => 0
    [bad_index_used] => 0
    [slow_queries] => 0
    [buffered_sets] => 0
    [unbuffered_sets] => 0
    [ps_buffered_sets] => 0
    [ps_unbuffered_sets] => 0
    [flushed_normal_sets] => 0
    [flushed_ps_sets] => 0
    [ps_prepared_never_executed] => 0
    [ps_prepared_once_executed] => 0
    [rows_fetched_from_server_normal] => 0
    [rows_fetched_from_server_ps] => 0
    [rows_buffered_from_client_normal] => 0
    [rows_buffered_from_client_ps] => 0
    [rows_fetched_from_client_normal_buffered] => 0
    [rows_fetched_from_client_normal_unbuffered] => 0
    [rows_fetched_from_client_ps_buffered] => 0
    [rows_fetched_from_client_ps_unbuffered] => 0
    [rows_fetched_from_client_ps_cursor] => 0
    [rows_skipped_normal] => 0
    [rows_skipped_ps] => 0
    [copy_on_write_saved] => 0
    [copy_on_write_performed] => 0
    [command_buffer_too_small] => 0
    [connect_success] => 1
    [connect_failure] => 0
    [connection_reused] => 0
    [reconnect] => 0
    [pconnect_success] => 0
    [active_connections] => 1
    [active_persistent_connections] => 0
    [explicit_close] => 0
    [implicit_close] => 0
    [disconnect_close] => 0
    [in_middle_of_command_close] => 0
    [explicit_free_result] => 0
    [implicit_free_result] => 0
    [explicit_stmt_close] => 0
    [implicit_stmt_close] => 0
    [mem_emalloc_count] => 0
    [mem_emalloc_ammount] => 0
    [mem_ecalloc_count] => 0
    [mem_ecalloc_ammount] => 0
    [mem_erealloc_count] => 0
    [mem_erealloc_ammount] => 0
    [mem_efree_count] => 0
    [mem_malloc_count] => 0
    [mem_malloc_ammount] => 0
    [mem_calloc_count] => 0
    [mem_calloc_ammount] => 0
    [mem_realloc_count] => 0
    [mem_realloc_ammount] => 0
    [mem_free_count] => 0
    [proto_text_fetched_null] => 0
    [proto_text_fetched_bit] => 0
    [proto_text_fetched_tinyint] => 0
    [proto_text_fetched_short] => 0
    [proto_text_fetched_int24] => 0
    [proto_text_fetched_int] => 0
    [proto_text_fetched_bigint] => 0
    [proto_text_fetched_decimal] => 0
    [proto_text_fetched_float] => 0
    [proto_text_fetched_double] => 0
    [proto_text_fetched_date] => 0
    [proto_text_fetched_year] => 0
    [proto_text_fetched_time] => 0
    [proto_text_fetched_datetime] => 0
    [proto_text_fetched_timestamp] => 0
    [proto_text_fetched_string] => 0
    [proto_text_fetched_blob] => 0
    [proto_text_fetched_enum] => 0
    [proto_text_fetched_set] => 0
    [proto_text_fetched_geometry] => 0
    [proto_text_fetched_other] => 0
    [proto_binary_fetched_null] => 0
    [proto_binary_fetched_bit] => 0
    [proto_binary_fetched_tinyint] => 0
    [proto_binary_fetched_short] => 0
    [proto_binary_fetched_int24] => 0
    [proto_binary_fetched_int] => 0
    [proto_binary_fetched_bigint] => 0
    [proto_binary_fetched_decimal] => 0
    [proto_binary_fetched_float] => 0
    [proto_binary_fetched_double] => 0
    [proto_binary_fetched_date] => 0
    [proto_binary_fetched_year] => 0
    [proto_binary_fetched_time] => 0
    [proto_binary_fetched_datetime] => 0
    [proto_binary_fetched_timestamp] => 0
    [proto_binary_fetched_string] => 0
    [proto_binary_fetched_blob] => 0
    [proto_binary_fetched_enum] => 0
    [proto_binary_fetched_set] => 0
    [proto_binary_fetched_geometry] => 0
    [proto_binary_fetched_other] => 0
)
```

See Also

Stats description

2.12.13. mysqli_get_client_stats

- `mysqli_get_client_stats`

  Returns client per-process statistics

Description

```
array mysqli_get_client_stats();
```

Warning

This function is currently not documented; only its argument list is available.

Returns client per-process statistics. Available only with mysqlnd.

Parameters

Return Values

Returns an array with client stats if success, `FALSE` otherwise.

Examples

Example 2.83. A `mysqli_get_client_stats` example

```
<?php
$link = mysqli_connect();
print_r(mysqli_get_client_stats());
?>
```

The above example will output something similar to:

```
Array
(
    [bytes_sent] => 43
    [bytes_received] => 80
    [packets_sent] => 1
    [packets_received] => 2
    [protocol_overhead_in] => 8
    [protocol_overhead_out] => 4
    [bytes_received_ok_packet] => 11
    [bytes_received_eof_packet] => 0
    [bytes_received_rset_header_packet] => 0
    [bytes_received_rset_field_meta_packet] => 0
    [bytes_received_rset_row_packet] => 0
    [bytes_received_prepare_response_packet] => 0
    [bytes_received_change_user_packet] => 0
    [packets_sent_command] => 0
    [packets_received_ok] => 1
    [packets_received_eof] => 0
    [packets_received_rset_header] => 0
    [packets_received_rset_field_meta] => 0
    [packets_received_rset_row] => 0
    [packets_received_prepare_response] => 0
    [packets_received_change_user] => 0
    [result_set_queries] => 0
    [non_result_set_queries] => 0
    [no_index_used] => 0
    [bad_index_used] => 0
    [slow_queries] => 0
    [buffered_sets] => 0
    [unbuffered_sets] => 0
    [ps_buffered_sets] => 0
    [ps_unbuffered_sets] => 0
    [flushed_normal_sets] => 0
    [flushed_ps_sets] => 0
    [ps_prepared_never_executed] => 0
    [ps_prepared_once_executed] => 0
    [rows_fetched_from_server_normal] => 0
    [rows_fetched_from_server_ps] => 0
    [rows_buffered_from_client_normal] => 0
    [rows_buffered_from_client_ps] => 0
    [rows_fetched_from_client_normal_buffered] => 0
    [rows_fetched_from_client_normal_unbuffered] => 0
    [rows_fetched_from_client_ps_buffered] => 0
    [rows_fetched_from_client_ps_unbuffered] => 0
    [rows_fetched_from_client_ps_cursor] => 0
    [rows_skipped_normal] => 0
    [rows_skipped_ps] => 0
    [copy_on_write_saved] => 0
    [copy_on_write_performed] => 0
    [command_buffer_too_small] => 0
    [connect_success] => 1
    [connect_failure] => 0
    [connection_reused] => 0
    [reconnect] => 0
    [pconnect_success] => 0
    [active_connections] => 1
    [active_persistent_connections] => 0
    [explicit_close] => 0
    [implicit_close] => 0
    [disconnect_close] => 0
    [in_middle_of_command_close] => 0
    [explicit_free_result] => 0
    [implicit_free_result] => 0
    [explicit_stmt_close] => 0
    [implicit_stmt_close] => 0
    [mem_emalloc_count] => 0
    [mem_emalloc_ammount] => 0
    [mem_ecalloc_count] => 0
    [mem_ecalloc_ammount] => 0
    [mem_erealloc_count] => 0
    [mem_erealloc_ammount] => 0
    [mem_efree_count] => 0
    [mem_malloc_count] => 0
    [mem_malloc_ammount] => 0
    [mem_calloc_count] => 0
    [mem_calloc_ammount] => 0
    [mem_realloc_count] => 0
    [mem_realloc_ammount] => 0
    [mem_free_count] => 0
    [proto_text_fetched_null] => 0
    [proto_text_fetched_bit] => 0
    [proto_text_fetched_tinyint] => 0
    [proto_text_fetched_short] => 0
    [proto_text_fetched_int24] => 0
    [proto_text_fetched_int] => 0
    [proto_text_fetched_bigint] => 0
    [proto_text_fetched_decimal] => 0
    [proto_text_fetched_float] => 0
    [proto_text_fetched_double] => 0
    [proto_text_fetched_date] => 0
    [proto_text_fetched_year] => 0
    [proto_text_fetched_time] => 0
    [proto_text_fetched_datetime] => 0
    [proto_text_fetched_timestamp] => 0
    [proto_text_fetched_string] => 0
    [proto_text_fetched_blob] => 0
    [proto_text_fetched_enum] => 0
    [proto_text_fetched_set] => 0
    [proto_text_fetched_geometry] => 0
    [proto_text_fetched_other] => 0
    [proto_binary_fetched_null] => 0
    [proto_binary_fetched_bit] => 0
    [proto_binary_fetched_tinyint] => 0
    [proto_binary_fetched_short] => 0
    [proto_binary_fetched_int24] => 0
    [proto_binary_fetched_int] => 0
    [proto_binary_fetched_bigint] => 0
    [proto_binary_fetched_decimal] => 0
    [proto_binary_fetched_float] => 0
    [proto_binary_fetched_double] => 0
    [proto_binary_fetched_date] => 0
    [proto_binary_fetched_year] => 0
    [proto_binary_fetched_time] => 0
    [proto_binary_fetched_datetime] => 0
    [proto_binary_fetched_timestamp] => 0
    [proto_binary_fetched_string] => 0
    [proto_binary_fetched_blob] => 0
    [proto_binary_fetched_enum] => 0
    [proto_binary_fetched_set] => 0
    [proto_binary_fetched_geometry] => 0
    [proto_binary_fetched_other] => 0
)
```

See Also

Stats description

2.12.14. mysqli_get_metadata

- mysqli_get_metadata

  Alias for mysqli_stmt_result_metadata

Description

This function is an alias of mysqli_stmt_result_metadata.

Notes

> **Note**
>
> mysqli_get_metadata is deprecated and will be removed.

See Also

mysqli_stmt_result_metadata

2.12.15. mysqli_master_query

- mysqli_master_query

  Enforce execution of a query on the master in a master/slave setup

Description

```
bool mysqli_master_query(mysqli link,
                         string query);
```

> **Warning**
>
> This function is currently not documented; only its argument list is available.

> **Warning**
>
> This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

2.12.16. mysqli_param_count

- mysqli_param_count

  Alias for mysqli_stmt_param_count

Description

This function is an alias of mysqli_stmt_param_count.

Notes

Note

mysqli_param_count is deprecated and will be removed.

See Also

mysqli_stmt_param_count

2.12.17. mysqli_report

- mysqli_report

  Enables or disables internal report functions

Description

```
bool mysqli_report(int flags);
```

mysqli_report is a powerful function to improve your queries and code during development and testing phase. Depending on the flags it reports errors from mysqli function calls or queries which don't use an index (or use a bad index).

Parameters

flags

Table 2.11. Supported flags

Name	Description
MYSQLI_REPORT_OFF	Turns reporting off
MYSQLI_REPORT_ERROR	Report errors from mysqli function calls
MYSQLI_REPORT_STRICT	Throw mysqli_sql_exception for errors instead of warnings
MYSQLI_REPORT_INDEX	Report if no index or bad index was used in a query
MYSQLI_REPORT_ALL	Set all options (report all)

Return Values

Returns TRUE on success or FALSE on failure.

Changelog

Version	Description
5.2.15 & 5.3.4	Changing the reporting mode is now be per-request, rather than per-process.

Examples

Example 2.84. Object oriented style

```
<?php
/* activate reporting */
mysqli_report(MYSQLI_REPORT_ALL);
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* this query should report an error */
$result = $mysqli->query("SELECT Name FROM Nonexistingtable WHERE population > 50000");
/* this query should report a bad index */
$result = $mysqli->query("SELECT Name FROM City WHERE population > 50000");
$result->close();
$mysqli->close();
?>
```

See Also

MySQLi_Driver::$report_mode
mysqli_debug
mysqli_dump_debug_info

2.12.18. mysqli_rpl_parse_enabled

- mysqli_rpl_parse_enabled

  Check if RPL parse is enabled

Description

```
int mysqli_rpl_parse_enabled(mysqli link);
```

> **Warning**
>
> This function is currently not documented; only its argument list is available.

> **Warning**
>
> This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

2.12.19. mysqli_rpl_probe

- mysqli_rpl_probe

  RPL probe

Description

```
bool mysqli_rpl_probe(mysqli link);
```

> **Warning**

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

2.12.20. mysqli_rpl_query_type, mysqli->rpl_query_type

- mysqli_rpl_query_type

  mysqli->rpl_query_type

  Returns RPL query type

Description

Object oriented style

```
 mysqli {

  int rpl_query_type(string query);

}
```

Procedural style

```
  int mysqli_rpl_query_type(mysqli link,
                            string query);
```

Returns MYSQLI_RPL_MASTER , MYSQLI_RPL_SLAVE or MYSQLI_RPL_ADMIN depending on a query type. INSERT, UPDATE and similar are *master* queries, SELECT is *slave*, and FLUSH, REPAIR and similar are *admin*.

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

2.12.21. mysqli_send_long_data

- mysqli_send_long_data

  Alias for mysqli_stmt_send_long_data

Description

This function is an alias of mysqli_stmt_send_long_data.

Notes

Note

mysqli_send_long_data is deprecated and will be removed.

See Also

mysqli_stmt_send_long_data

2.12.22. mysqli_send_query, mysqli->send_query

- mysqli_send_query

  mysqli->send_query

  Send the query and return

Description

Object oriented style

```
 mysqli {
  bool send_query(string query);
}
```

Procedural style

```
  bool mysqli_send_query(mysqli link,
                         string query);
```

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

2.12.23. mysqli_set_opt

- mysqli_set_opt

  Alias of mysqli_options

Description

This function is an alias of mysqli_options.

2.12.24. mysqli_slave_query

- mysqli_slave_query

  Force execution of a query on a slave in a master/slave setup

Description

```
  bool mysqli_slave_query(mysqli link,
                          string query);
```

> **Warning**
>
> This function is currently not documented; only its argument list is available.

> **Warning**
>
> This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

Chapter 3. MySQL Native Driver (`Mysqlnd`)

This section of the manual provides an overview of the MySQL Native Driver.

3.1. Overview

MySQL Native Driver is a replacement for the MySQL Client Library (libmysql). MySQL Native Driver is part of the official PHP sources as of PHP 5.3.0.

The MySQL database extensions MySQL extension, `mysqli` and PDO MYSQL all communicate with the MySQL server. In the past, this was done by the extension using the services provided by the MySQL Client Library. The extensions were compiled against the MySQL Client Library in order to use its client-server protocol.

With MySQL Native Driver there is now an alternative, as the MySQL database extensions can be compiled to use MySQL Native Driver instead of the MySQL Client Library.

MySQL Native Driver is written in C as a PHP extension.

What it is not

Although MySQL Native Driver is written as a PHP extension, it is important to note that it does not provide a new API to the PHP programmer. The programmer APIs for MySQL database connectivity are provided by the MySQL extension, `mysqli` and PDO MYSQL. These extensions can now use the services of MySQL Native Driver to communicate with the MySQL Server. Therefore, you should not think of MySQL Native Driver as an API.

Why use it?

Using the MySQL Native Driver offers a number of advantages over using the MySQL Client Library.

Because MySQL Native Driver is written as a PHP extension, it is tightly coupled to the workings of PHP. This leads to gains in efficiency, especially when it comes to memory usage, as the driver uses the PHP memory management system. It also supports the PHP memory limit. Using MySQL Native Driver leads to comparable or better performance than using MySQL Client Library, it always ensures the most efficient use of memory. One example of the memory efficiency is the fact that when using the MySQL Client Library, each row is stored in memory twice, whereas with the MySQL Native Driver each row is only stored once in memory.

Also, in the past, you needed to build the MySQL database extensions against a copy of the MySQL Client Library. This typically meant you needed to have MySQL installed on a machine where you were building the PHP source code. Also, when your PHP application was running, the MySQL database extensions would call down to the MySQL Client library file at run time, so the file needed to be installed on your system. With MySQL Native Driver that is no longer the case as it is included as part of the standard distribution. So you do not need MySQL installed in order to build PHP or run PHP database applications.

With MySQL Native Driver it is also possible to write plugins. These are PHP extensions written to use the MySQL Native Driver Plugin API, and they can extend the functionality of the driver without requiring changes to application level code.

The older MySQL Client Library was written by MySQL AB (now Oracle Corporation) and so was released under the MySQL license. This ultimately led to MySQL support being disabled by default in PHP. However, the MySQL Native Driver has been developed as part of the PHP project, and is therefore released under the PHP license. This removes licensing issues that have been problematic in the past.

Special features

MySQL Native Driver also provides some special features not available when the MySQL database extensions use MySQL Client Library. These special features are listed below:

- Improved persistent connections
- The special function `mysqli_fetch_all`
- Performance statistics calls: `mysqli_get_cache_stats`, `mysqli_get_client_stats`,

`mysqli_get_connection_stats`

- Asynchronous queries: `mysqli_poll`, `mysqli_reap_async_query`

The performance statistics facility can prove to be very useful in identifying performance bottlenecks.

MySQL Native Driver also allows for persistent connections when used with the `mysqli` extension.

SSL Support

MySQL Native Driver supports SSL from PHP version 5.3.3.

Compressed Protocol Support

As of PHP 5.3.2 MySQL Native Driver supports the compressed client server protocol. MySQL Native Driver did not support this in 5.3.0 and 5.3.1. Extensions such as `ext/mysql`, `ext/mysqli`, `PDO_MYSQL`, that are configured to use MySQL Native Driver, can also take advantage of this feature.

3.2. Installation

Installation on Unix

By default the MySQL database extensions are configured to use MySQL Client Library. In order to use the MySQL Native Driver, PHP needs to be built specifying that the MySQL database extensions are compiled with MySQL Native Driver support. This is done through configuration options prior to building the PHP source code.

For example, to build the MySQL extension, `mysqli` and PDO MYSQL using the MySQL Native Driver, the following command would be given:

```
./configure --with-mysql=mysqlnd \
--with-mysqli=mysqlnd \
--with-pdo-mysql=mysqlnd \
[other options]
```

Installation on Windows

In the official PHP distributions from 5.3 onwards, MySQL Native Driver is enabled by default, so no additional configuration is required to use it. All MySQL database extensions will use MySQL Native Driver in this case.

3.3. Runtime Configuration

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

Table 3.1. MySQL Native Driver Configuration Options

Name	Default	Changeable	Changelog
mysqlnd.collect_statistics	"1"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqlnd.collect_memory_statistics	"0"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqlnd.log_mask	"0"	PHP_INI_SYSTEM	Available since PHP 5.3.0.

For further details and definitions of the PHP_INI_* modes, see the configuration.changes.modes.

Here's a short explanation of the configuration directives.

mysqlnd.collect_statistics boolean

Enables the collection of various client statistics which can be accessed through mysqli_get_client_stats, mysqli_get_connection_stats, mysqli_get_cache_stats and are shown in mysqlnd section of the output of the phpinfo function as well.

This configuration setting enables all MySQL Native Driver statistics except those relating to memory management.

mysqlnd.collect_memory_statistics boolean

Enable the collection of various memory statistics which can be accessed through mysqli_get_client_stats, mysqli_get_connection_stats, mysqli_get_cache_stats and are shown in mysqlnd section of the output of the phpinfo function as well.

This configuration setting enables the memory management statistics within the overall set of MySQL Native Driver statistics.

mysqlnd.log_mask integer

The PHP configuration setting mysqlnd.log_mask defines which queries will be logged. The default is mysqlnd.log_mask = "0", which disables logging.

This setting can have values assigned according to the following table:

SERVER_STATUS_IN_TRANS	1
SERVER_STATUS_AUTOCOMMIT	2
SERVER_MORE_RESULTS_EXISTS	8
SERVER_QUERY_NO_GOOD_INDEX_USED	16
SERVER_QUERY_NO_INDEX_USED	32
SERVER_STATUS_CURSOR_EXISTS	64
SERVER_STATUS_LAST_ROW_SENT	128
SERVER_STATUS_DB_DROPPED	256
SERVER_STATUS_NO_BACKSLASH_ESCAPES	512
SERVER_QUERY_WAS_SLOW	1024

It is possible to add the values together, to log multiple query types. For example, you can set it to 16 + 32 = 48 to log slow queries which either use no good index (SERVER_QUERY_NO_GOOD_INDEX_USED = 16) or no index at all (SERVER_QUERY_NO_INDEX_USED = 32).

Note, the SERVER_* constants are MySQL internal (C-level) constants, not exported to the PHP userland. There are no corresponding PHP constants. However, you can look up the values in the table which has been created by inspecting the MySQL Native Driver source.

3.4. Persistent Connections

Using Persistent Connections

If mysqli is used with mysqlnd, when a persistent connection is created it generates a COM_CHANGE_USER (mysql_change_user()) call on the server. This ensures that re-authentication of the connection takes place.

As there is some overhead associated with the COM_CHANGE_USER call, it is possible to switch this off at compile time. Reusing a persistent connection will then generate a COM_PING (mysql_ping) call to simply test the connection is reusable.

Generation of COM_CHANGE_USER can be switched off with the compile flag MYSQLI_NO_CHANGE_USER_ON_PCONNECT. For example:

```
shell# CFLAGS="-DMYSQLI_NO_CHANGE_USER_ON_PCONNECT" ./configure --with-mysql=/usr/local/mysql/ --with-mysqli=/usr/local/my
```

Or alternatively:

```
shell# export CFLAGS="-DMYSQLI_NO_CHANGE_USER_ON_PCONNECT"
shell# configure --whatever-option
shell# make clean
shell# make
```

Note that only `mysqli` on `mysqlnd` uses `COM_CHANGE_USER`. Other extension-driver combinations use `COM_PING` on initial use of a persistent connection.

3.5. Statistics

Using Statistical Data

MySQL Native Driver contains support for gathering statistics on the communication between the client and the server. The statistics gathered are of three main types:

- Client statistics
- Connection statistics
- Zval cache statistics

If you are using the `mysqli` extension, these statistics can be obtained through three API calls:

- `mysqli_get_client_stats`
- `mysqli_get_connection_stats`
- `mysqli_get_cache_stats`

> **Note**
>
> Statistics are aggregated among all extensions that use MySQL Native Driver. For example, when compiling both `ext/mysql` and `ext/mysqli` against MySQL Native Driver, both function calls of `ext/mysql` and `ext/mysqli` will change the statistics. There is no way to find out how much a certain API call of any extension that has been compiled against MySQL Native Driver has impacted a certain statistic. You can configure the PDO MySQL Driver, `ext/mysql` and `ext/mysqli` to optionally use the MySQL Native Driver. When doing so, all three extensions will change the statistics.

Accessing Client Statistics

To access client statistics, you need to call `mysqli_get_client_stats`. The function call does not require any parameters.

The function returns an associative array that contains the name of the statistic as the key and the statistical data as the value.

Client statistics can also be accessed by calling the `phpinfo` function.

Accessing Connection Statistics

To access connection statistics call `mysqli_get_connection_stats`. This takes the database connection handle as the parameter.

The function returns an associative array that contains the name of the statistic as the key and the statistical data as the value.

Accessing Zval Cache Statistics

The MySQL Native Driver also collects statistics from its internal Zval cache. These statistics can be accessed by calling `mysqli_get_cache_stats`.

The Zval cache statistics obtained may lead to a tweaking of php.ini settings related to the Zval cache, resulting in better performance.

Buffered and Unbuffered Result Sets

Result sets can be buffered or unbuffered. Using default settings, ext/mysql and ext/mysqli work with buffered result sets for normal (non prepared statement) queries. Buffered result sets are cached on the client. After the query execution all results are fetched from the MySQL Server and stored in a cache on the client. The big advantage of buffered result sets is that they allow the server to free all resources allocated to a result set, once the results have been fetched by the client.

Unbuffered result sets on the other hand are kept much longer on the server. If you want to reduce memory consumption on the client, but increase load on the server, use unbuffered results. If you experience a high server load and the figures for unbuffered result sets are high, you should consider moving the load to the clients. Clients typically scale better than servers. “Load” does not only refer to memory buffers - the server also needs to keep other resources open, for example file handles and threads, before a result set can be freed.

Prepared Statements use unbuffered result sets by default. However, you can use mysqli_stmt_store_result to enable buffered result sets.

Statistics returned by MySQL Native Driver

The following tables show a list of statistics returned by the mysqli_get_client_stats, mysqli_get_connection_stats and mysqli_get_cache_stats functions.

Network

Statistic	**Scope**	**Description**	**Notes**
bytes_sent	Connection	Number of bytes sent from PHP to the MySQL server	Can be used to check the efficiency of the compression protocol
bytes_received	Connection	Number of bytes received from MySQL server	Can be used to check the efficiency of the compression protocol
packets_sent	Connection	Number of MySQL Client Server protocol packets sent	Used for debugging Client Server protocol implementation
packets_received	Connection	Number of MySQL Client Server protocol packets received	Used for debugging Client Server protocol implementation
protocol_overhead_in	Connection	MySQL Client Server protocol overhead in bytes for incoming traffic. Currently only the Packet Header (4 bytes) is considered as overhead. protocol_overhead_in = packets_received * 4	Used for debugging Client Server protocol implementation
protocol_overhead_out	Connection	MySQL Client Server protocol overhead in bytes for outgoing traffic. Currently only the Packet Header (4 bytes) is considered as overhead. protocol_overhead_out = packets_sent * 4	Used for debugging Client Server protocol implementation
bytes_received_ok_packet	Connection	Total size of bytes of MySQL Client Server protocol OK packets received. OK packets can contain a status message. The length of the status message can vary and thus the size of an OK packet is not fixed.	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
packets_received_ok	Connection	Number of MySQL Client Server protocol OK packets received.	Used for debugging CS protocol implementation. Note that the total size in bytes includes the

Statistic	Scope	Description	Notes
			size of the header packet (4 bytes, see protocol overhead).
bytes_received_eof_packet	Connection	Total size in bytes of MySQL Client Server protocol EOF packets received. EOF can vary in size depending on the server version. Also, EOF can transport an error message.	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
packets_received_eof	Connection	Number of MySQL Client Server protocol EOF packets. Like with other packet statistics the number of packets will be increased even if PHP does not receive the expected packet but, for example, an error message.	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
bytes_received_rset_header_packet	Connection	Total size in bytes of MySQL Client Server protocol result set header packets. The size of the packets varies depending on the payload (LOAD LOCAL INFILE, INSERT, UPDATE, SELECT, error message).	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
packets_received_rset_header	Connection	Number of MySQL Client Server protocol result set header packets.	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
bytes_received_rset_field_meta_packet	Connection	Total size in bytes of MySQL Client Server protocol result set meta data (field information) packets. Of course the size varies with the fields in the result set. The packet may also transport an error or an EOF packet in case of COM_LIST_FIELDS.	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
packets_received_rset_field_meta	Connection	Number of MySQL Client Server protocol result set meta data (field information) packets.	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
bytes_received_rset_row_packet	Connection	Total size in bytes of MySQL Client Server protocol result set row data packets. The packet may also transport an error or an EOF packet. You can reverse engineer the number of error and EOF packets by substracting rows_fetched_from_server_normal and rows_fetched_from_server_ps from bytes_received_rset_row_packet.	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
packets_received_rset_row	Connection	Number of MySQL Client Server protocol result set row data packets and their total size in bytes.	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol

Statistic	Scope	Description	Notes
			overhead).
bytes_received_prepare_response_packet	Connection	Total size in bytes of MySQL Client Server protocol OK for Prepared Statement Initialization packets (prepared statement init packets). The packet may also transport an error. The packet size depends on the MySQL version: 9 bytes with MySQL 4.1 and 12 bytes from MySQL 5.0 on. There is no safe way to know how many errors happened. You may be able to guess that an error has occurred if, for example, you always connect to MySQL 5.0 or newer and, bytes_received_prepare_response_packet != packets_received_prepare_response * 12. See also ps_prepared_never_executed, ps_prepared_once_executed.	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
packets_received_prepare_response	Connection	Number of MySQL Client Server protocol OK for Prepared Statement Initialization packets (prepared statement init packets).	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
bytes_received_change_user_packet	Connection	Total size in bytes of MySQL Client Server protocol COM_CHANGE_USER packets. The packet may also transport an error or EOF.	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
packets_received_change_user	Connection	Number of MySQL Client Server protocol COM_CHANGE_USER packets	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
packets_sent_command	Connection	Number of MySQL Client Server protocol commands sent from PHP to MySQL. There is no way to know which specific commands and how many of them have been sent. At its best you can use it to check if PHP has sent any commands to MySQL to know if you can consider to disable MySQL support in your PHP binary. There is also no way to reverse engineer the number of errors that may have occurred while sending data to MySQL. The only error recoded is com-	Only useful for debugging CS protocol implementation.

Statistic	Scope	Description	Notes
		mand_buffer_too_small (see below).	

Result Set

Statistic	Scope	Description	Notes
result_set_queries	Connection	Number of queries that have generated a result set. Examples of queries that generate a result set: SELECT, SHOW. The statistic will not be incremented if there is an error reading the result set header packet from the line.	You may use it as an indirect measure for the number of queries PHP has sent to MySQL, for example, to identify a client that causes a high database load.
non_result_set_queries	Connection	Number of queries that did not generate a result set. Examples of queries that do not generate a result set: INSERT, UPDATE, LOAD DATA, SHOW. The statistic will not be incremented if there is an error reading the result set header packet from the line.	You may use it an an indirect measure for the number of queries PHP has sent to MySQL, for example, to identify a client that causes a high database load.
no_index_used	Connection	Number of queries that have generated a result set but did not use an index (see also mysqld start option –log-queries-not-using-indexes). If you want these queries to be reported you can use mysqli_report(MYSQLI_REPORT_INDEX) to make ext/mysqli throw an exception. If you prefer a warning instead of an exception use mysqli_report(MYSQLI_REPORT_INDEX ^ MYSQLI_REPORT_STRICT).	
bad_index_used	Connection	Number of queries that have generated a result set and did not use a good index (see also mysqld start option –log-slow-queries).	If you want these queries to be reported you can use mysqli_report(MYSQLI_REPORT_INDEX) to make ext/mysqli throw an exception. If you prefer a warning instead of an exception use mysqli_report(MYSQLI_REPORT_INDEX ^ MYSQLI_REPORT_STRICT)
slow_queries	Connection	SQL statements that took more than long_query_time seconds to execute and required at least min_examined_row_limit rows to be examined.	Not reported through mysqli_report
buffered_sets	Connection	Number of buffered result sets returned by “normal” queries. “Normal” means “not prepared statement” in the following notes.	Examples of API calls that will buffer result sets on the client: mysql_query, mysqli_query, mysqli_store_result,

Statistic	Scope	Description	Notes

Statistic	Scope	Description	Notes
			. Buffering result sets on the client ensures that server resources are freed as soon as possible and it makes result set scrolling easier. The downside is the additional memory consumption on the client for buffering data. Note that mysqlnd (unlike the MySQL Client Library) respects the PHP memory limit because it uses PHP internal memory management functions to allocate memory. This is also the reason why `memory_get_usage` reports a higer memory consumption when using mysqlnd instead of the MySQL Client Library. `memory_get_usage` does not measure the memory consumption of the MySQL Client Library at all because the MySQL Client Library does not use PHP internal memory management functions monitored by the function!
`unbuffered_sets`	Connection	Number of unbuffered result sets returned by normal (non prepared statement) queries.	Examples of API calls that will not buffer result sets on the client: `mysqli_use_result`
`ps_buffered_sets`	Connection	Number of buffered result sets returned by prepared statements. By default prepared statements are unbuffered.	Examples of API calls that will not buffer result sets on the client: `mysqli_stmt_store_result`
`ps_unbuffered_sets`	Connection	Number of unbuffered result sets returned by prepared statements.	By default prepared statements are unbuffered.
`flushed_normal_sets`	Connection	Number of result sets from normal (non prepared statement) queries with unread data which have been flushed silently for you. Flushing happens only with unbuffered result sets.	Unbuffered result sets must be fetched completely before a new query can be run on the connection otherwise MySQL will throw an error. If the application does not fetch all rows from an unbuffered result set, mysqlnd does implicitly fetch the result set to clear the line. See also `rows_skipped_normal`, `rows_skipped_ps`. Some possible causes for an implicit flush: • Faulty client application • Client stopped reading after it found what it was looking for but has made MySQL calculate more records than needed • Client application has

Statistic	Scope	Description	Notes
			stopped unexpectedly
flushed_ps_sets	Connection	Number of result sets from prepared statements with unread data which have been flushed silently for you. Flushing happens only with unbuffered result sets.	Unbuffered result sets must be fetched completely before a new query can be run on the connection otherwise MySQL will throw an error. If the application does not fetch all rows from an unbuffered result set, mysqlnd does implicitly fetch the result set to clear the line. See also rows_skipped_normal, rows_skipped_ps. Some possible causes for an implicit flush: • Faulty client application • Client stopped reading after it found what it was looking for but has made MySQL calculate more records than needed • Client application has stopped unexpectedly
ps_prepared_never_executed	Connection	Number of statements prepared but never executed.	Prepared statements occupy server resources. You should not prepare a statement if you do not plan to execute it.
ps_prepared_once_executed	Connection	Number of prepared statements executed only one.	One of the ideas behind prepared statements is that the same query gets executed over and over again (with different parameters) and some parsing and other preparation work can be saved, if statement execution is split up in separate prepare and execute stages. The idea is to prepare once and "cache" results, for example, the parse tree to be reused during multiple statement executions. If you execute a prepared statement only once the two stage processing can be inefficient compared to "normal" queries because all the caching means extra work and it takes (limited) server resources to hold the cached information. Consequently, prepared statements that are executed only once may cause performance hurts.
rows_fetched_from_server_normal, rows_fetched_from_server_ps	Connection	Total number of result set rows successfully fetched from MySQL regardless if the client application has consumed them	See also packets_received_rset_row

Statistic	Scope	Description	Notes
		or not. Some of the rows may not have been fetched by the client application but have been flushed implicitly.	
rows_buffered_from_client_normal, rows_buffered_from_client_ps	Connection	Total number of succesfully buffered rows originating from a "normal" query or a prepared statement. This is the number of rows that have been fetched from MySQL and buffered on client. Note that there are two distinct statistics on rows that have been buffered (MySQL to mysqlnd internal buffer) and buffered rows that have been fetched by the client application (mysqlnd internal buffer to client application). If the number of buffered rows is higher than the number of fetched buffered rows it can mean that the client application runs queries that cause larger result sets than needed resulting in rows not read by the client.	Examples of queries that will buffer results: mysqli_query, mysqli_store_result
rows_fetched_from_client_normal_buffered, rows_fetched_from_client_ps_buffered	Connection	Total number of rows fetched by the client from a buffered result set created by a normal query or a prepared statement.	
rows_fetched_from_client_normal_unbuffered, rows_fetched_from_client_ps_unbuffered	Connection	Total number of rows fetched by the client from a unbuffered result set created by a "normal" query or a prepared statement.	
rows_fetched_from_client_ps_cursor	Connection	Total number of rows fetch by the client from a cursor created by a prepared statement.	
rows_skipped_normal, rows_skipped_ps	Connection	Reserved for future use (currently not supported)	
copy_on_write_saved, copy_on_write_performed	Process	With mysqlnd, variables returned by the extensions point into mysqlnd internal network result buffers. If you do not change the variables, fetched data will be kept only once in memory. If you change the variables, mysqlnd has to perform a copy-on-write to protect the internal network result buffers from being changed. With the MySQL Client Library you always hold fetched data twice in memory. Once in the internal MySQL Client Library buffers and once in the variables returned by the extensions. In theory mysqlnd can save up to 40% memory. However, note that the memory saving cannot be measured using memory_get_usage.	

Statistic	Scope	Description	Notes
explicit_free_result, implicit_free_result	Connection, Process (only during prepared statement cleanup)	Total number of freed result sets.	The free is always considered explicit but for result sets created by an init command, for example, mysqli_options(MYSQLI_INIT_COMMAND , ...)
proto_text_fetched_null, proto_text_fetched_bit, proto_text_fetched_tinyint proto_text_fetched_short, proto_text_fetched_int24, proto_text_fetched_int proto_text_fetched_bigint, proto_text_fetched_decimal, proto_text_fetched_float proto_text_fetched_double, proto_text_fetched_date, proto_text_fetched_year proto_text_fetched_time, proto_text_fetched_datetime, proto_text_fetched_timestamp proto_text_fetched_string, proto_text_fetched_blob, proto_text_fetched_enum proto_text_fetched_set, proto_text_fetched_geometry, proto_text_fetched_other	Connection	Total number of columns of a certain type fetched from a normal query (MySQL text protocol).	Mapping from C API / MySQL meta data type to statistics name: • MYSQL_TYPE_NULL - proto_text_fetched_null • MYSQL_TYPE_BIT - proto_text_fetched_bit • MYSQL_TYPE_TINY - proto_text_fetched_tinyint • MYSQL_TYPE_SHORT - proto_text_fetched_short • MYSQL_TYPE_INT24 - proto_text_fetched_int24 • MYSQL_TYPE_LONG - proto_text_fetched_int • MYSQL_TYPE_LONGLONG - proto_text_fetched_bigint • MYSQL_TYPE_DECIMAL, MYSQL_TYPE_NEWDECIMAL - proto_text_fetched_decimal • MYSQL_TYPE_FLOAT - proto_text_fetched_float • MYSQL_TYPE_DOUBLE - proto_text_fetched_double • MYSQL_TYPE_DATE, MYSQL_TYPE_NEWDATE - proto_text_fetched_date • MYSQL_TYPE_YEAR - proto_text_fetched_year • MYSQL_TYPE_TIME - proto_text_fetched_time • MYSQL_TYPE_DATETIME - proto_text_fetched_datetime • MYSQL_TYPE_TIMESTAMP - proto_text_fetched_timestamp • MYSQL_TYPE_STRING, MYSQL_TYPE_VARSTRIN

Statistic	Scope	Description	Notes
			G, MYSQL_TYPE_VARCHAR - proto_text_fetched_string • MYSQL_TYPE_TINY_BLOB, MYSQL_TYPE_MEDIUM_BLOB, MYSQL_TYPE_LONG_BLOB, MYSQL_TYPE_BLOB - proto_text_fetched_blob • MYSQL_TYPE_ENUM - proto_text_fetched_enum • MYSQL_TYPE_SET - proto_text_fetched_set • MYSQL_TYPE_GEOMETRY - proto_text_fetched_geometry • Any MYSQL_TYPE_* not listed before (there should be none) - proto_text_fetched_other Note that the MYSQL_*-type constants may not be associated with the very same SQL column types in every version of MySQL.
proto_binary_fetched_null, proto_binary_fetched_bit, proto_binary_fetched_tinyint proto_binary_fetched_short, proto_binary_fetched_int24, proto_binary_fetched_int, proto_binary_fetched_bigint, proto_binary_fetched_decimal, proto_binary_fetched_float, proto_binary_fetched_double, proto_binary_fetched_date, proto_binary_fetched_year, proto_binary_fetched_time, proto_binary_fetched_datetime,	Connection	Total number of columns of a certain type fetched from a prepared statement (MySQL binary protocol).	For type mapping see proto_text_* described in the preceding text.

Statistic	Scope	Description	Notes
proto_binary_fetched_timestamp, proto_binary_fetched_string, proto_binary_fetched_blob, proto_binary_fetched_enum, proto_binary_fetched_set, proto_binary_fetched_geometry, proto_binary_fetched_other			

Connection

Statistic	Scope	Description	Notes
connect_success, connect_failure	Connection	Total number of successful / failed connection attempt.	Reused connections and all other kinds of connections are included.
reconnect	Process	Total number of (real_)connect attempts made on an already opened connection handle.	The code sequence `$link = new mysqli(...); $link->real_connect(...)` will cause a reconnect. But `$link = new mysqli(...); $link->connect(...)` will not because `$link->connect(...)` will explicitly close the existing connection before a new connection is established.
pconnect_success	Connection	Total number of successful persistent connection attempts.	Note that connect_success holds the sum of successful persistent and non-persistent connection attempts. The number of successful non-persistent connection attempts is connect_success - pconnect_success.
active_connections	Connection	Total number of active persistent and non-persistent connections.	
active_persistent_connections	Connection	Total number of active persistent connections.	The total number of active non-persistent connections is active_connections - active_persistent_connections.

explicit_close	Connection	Total number of explicitly closed connections (ext/mysqli only).	Examples of code snippets that cause an explicit close :
			`$link = new mysqli(...); $link->close`
			`$link = new mysqli(...); $link->conne`

implicit_close	Connection	Total number of implicitly closed connections (ext/mysqli only).	Examples of code snippets that cause an implicit close :

Statistic	Scope	Description	Notes
			• `$link = new mysqli(...); $link->real_connect(...)` • `unset($link)` • Persistent connection: pooled connection has been created with real_connect and there may be unknown options set - close implicitly to avoid returning a connection with unknown options • Persistent connection: ping/change_user fails and ext/mysqli closes the connection • end of script execution: close connections that have not been closed by the user
`disconnect_close`	Connection	Connection failures indicated by the C API call `mysql_real_connect` during an attempt to establish a connection.	It is called `disconnect_close` because the connection handle passed to the C API call will be closed.
`in_middle_of_command_close`	Process	A connection has been closed in the middle of a command execution (outstanding result sets not fetched, after sending a query and before retrieving an answer, while fetching data, while transferring data with LOAD DATA).	Unless you use asynchronous queries this should only happen if your script stops unexpectedly and PHP shuts down the connections for you.
`init_command_executed_count`	Connection	Total number of init command executions, for example, `mysqli_options(MYSQLI_INIT_COMMAND , ...)`.	The number of successful executions is `init_command_executed_count` - `init_command_failed_count`.
`init_command_failed_count`	Connection	Total number of failed init commands.	

COM_* Commands

Statistic	Scope	Description	Notes
`com_quit`, `com_init_db`, `com_query`, `com_field_list`, `com_create_db`, `com_drop_db`, `com_refresh`, `com_shutdown`, `com_statistics`, `com_process_info`, `com_connect`, `com_process_kill`,	Connection	Total number of attempts to send a certain COM_* command from PHP to MySQL.	The statistics are incremented after checking the line and immediately before sending the corresponding MySQL client server protocol packet. If mysqlnd fails to send the packet over the wire the statistics will not be decremented. In case of a failure mysqlnd emits a PHP warning “Error while sending

Statistic	Scope	Description	Notes
com_debug, com_ping, com_time, com_delayed_insert, com_change_user, com_binlog_dump, com_table_dump, com_connect_out, com_register_slave, com_stmt_prepare, com_stmt_execute, com_stmt_send_long_data, com_stmt_close, com_stmt_reset, com_stmt_set_option, com_stmt_fetch, com_daemon			%s packet. PID=%d." Usage examples: • Check if PHP sends certain commands to MySQL, for example, check if a client sends COM_PROCESS_KILL • Calculate the average number of prepared statement executions by comparing COM_EXECUTE with COM_PREPARE • Check if PHP has run any non-prepared SQL statements by checking if COM_QUERY is zero • Identify PHP scripts that run an excessive number of SQL statements by checking COM_QUERY and COM_EXECUTE

Miscellaneous

Statistic	Scope	Description	Notes
explicit_stmt_close, implicit_stmt_close	Process	Total number of close prepared statements.	A close is always considered explicit but for a failed prepare.
mem_emalloc_count, mem_emalloc_ammount, mem_ecalloc_count, mem_ecalloc_ammount, mem_erealloc_count, mem_erealloc_ammount, mem_efree_count, mem_malloc_count, mem_malloc_ammount, mem_calloc_count, mem_calloc_ammount, mem_realloc_count, mem_realloc_ammount, mem_free_count	Process	Memory management calls.	Development only.
command_buffer_too_small	Connection	Number of network command buffer extensions while sending commands from PHP to MySQL.	mysqlnd allocates an internal command/network buffer of mysqlnd.net_cmd_buffer_size (php.ini) bytes for every connection. If a MySQL Client Server protocol command, for example, COM_QUERY (normal query), does not fit into the buffer, mysqlnd will grow the buffer to what is needed for sending the

Statistic	Scope	Description	Notes
			command. Whenever the buffer gets extended for one connection command_buffer_too_small will be incremented by one. If mysqlnd has to grow the buffer beyond its initial size of mysqlnd.net_cmd_buffer_size (php.ini) bytes for almost every connection, you should consider to increase the default size to avoid re-allocations. The default buffer size is 2048 bytes in PHP 5.3.0. In future versions the default will be 4kB or larger. The default can changed either through the php.ini setting mysqlnd.net_cmd_buffer_size or using mysqli_options(MYSQLI_OPT_NET_CMD_BUFFER_SIZE, int size). It is recommended to set the buffer size to no less than 4096 bytes because mysqlnd also uses it when reading certain communication packet from MySQL. In PHP 5.3.0, mysqlnd will not grow the buffer if MySQL sends a packet that is larger than the current size of the buffer. As a consequence mysqlnd is unable to decode the packet and the client application will get an error. There are only two situations when the packet can be larger than the 2048 bytes default of mysqlnd.net_cmd_buffer_size in PHP 5.3.0: the packet transports a very long error message or the packet holds column meta data from COM_LIST_FIELD (mysql_list_fields) and the meta data comes from a string column with a very long default value (>1900 bytes). No bug report on this exists - it should happen rarely. As of PHP 5.3.2 mysqlnd does not allow setting buffers smaller than 4096 bytes.
connection_reused			

3.6. Notes

This section provides a collection of miscellaneous notes on MySQL Native Driver usage.

- Using `mysqlnd` means using PHP streams for underlying connectivity. For `mysqlnd`, the PHP streams documentation (book.stream) should be consulted on such details as timeout settings, not the documentation for the MySQL Client Library.

Chapter 4. MySQL Functions (PDO_MYSQL)

PDO_MYSQL is a driver that implements the PHP Data Objects (PDO) interface to enable access from PHP to MySQL 3.x, 4.x and 5.x databases.

PDO_MYSQL will take advantage of native prepared statement support present in MySQL 4.1 and higher. If you're using an older version of the mysql client libraries, PDO will emulate them for you.

Warning

Beware: Some MySQL table types (storage engines) do not support transactions. When writing transactional database code using a table type that does not support transactions, MySQL will pretend that a transaction was initiated successfully. In addition, any DDL queries issued will implicitly commit any pending transactions.

The constants below are defined by this driver, and will only be available when the extension has been either compiled into PHP or dynamically loaded at runtime. In addition, these driver-specific constants should only be used if you are using this driver. Using driver-specific attributes with another driver may result in unexpected behaviour. `PDO::getAttribute` may be used to obtain the `PDO_ATTR_DRIVER_NAME` attribute to check the driver, if your code can run against multiple drivers.

`PDO::MYSQL_ATTR_USE_BUFFERED_QUERY` (integer)
: If this attribute is set to `TRUE` on a `PDOStatement`, the MySQL driver will use the buffered versions of the MySQL API. If you're writing portable code, you should use `PDOStatement::fetchAll` instead.

Example 4.1. Forcing queries to be buffered in mysql

```
<?php
if ($db->getAttribute(PDO::ATTR_DRIVER_NAME) == 'mysql') {
    $stmt = $db->prepare('select * from foo',
        array(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY => true));
} else {
    die("my application only works with mysql; I should use \$stmt->fetchAll() instead");
}
?>
```

`PDO::MYSQL_ATTR_LOCAL_INFILE` (integer)
: Enable `LOAD LOCAL INFILE`.

`PDO::MYSQL_ATTR_INIT_COMMAND` (integer)
: Command to execute when connecting to the MySQL server. Will automatically be re-executed when reconnecting.

`PDO::MYSQL_ATTR_READ_DEFAULT_FILE` (integer)
: Read options from the named option file instead of from `my.cnf`. This option is not available if mysqlnd is used, because mysqlnd does not read the mysql configuration files.

`PDO::MYSQL_ATTR_READ_DEFAULT_GROUP` (integer)
: Read options from the named group from `my.cnf` or the file specified with `MYSQL_READ_DEFAULT_FILE`. This option is not available if mysqlnd is used, because mysqlnd does not read the mysql configuration files.

`PDO::MYSQL_ATTR_MAX_BUFFER_SIZE` (integer)
: Maximum buffer size. Defaults to 1 MiB.

`PDO::MYSQL_ATTR_DIRECT_QUERY` (integer)
: Perform direct queries, don't use prepared statements.

4.1. `PDO_MYSQL DSN`

- `PDO_MYSQL DSN`

Connecting to MySQL databases

Description

The PDO_MYSQL Data Source Name (DSN) is composed of the following elements:

DSN prefix	The DSN prefix is **`mysql:`**.
`host`	The hostname on which the database server resides.
`port`	The port number where the database server is listening.
`dbname`	The name of the database.
`unix_socket`	The MySQL Unix socket (shouldn't be used with `host` or `port`).

Examples

Example 4.2. PDO_MYSQL DSN examples

The following example shows a PDO_MYSQL DSN for connecting to MySQL databases:

```
mysql:host=localhost;dbname=testdb
```

More complete examples:

```
mysql:host=localhost;port=3307;dbname=testdb
mysql:unix_socket=/tmp/mysql.sock;dbname=testdb
```

Chapter 5. Connector/PHP

The MySQL Connector/PHP is a version of the `mysql` and `mysqli` extensions for PHP optimized for the Windows operating system. Later versions of the main PHP `mysql`/`mysqli` drivers are compatible with Windows and a separate, Windows specific driver is no longer required.

For PHP for all platforms, including Windows, you should use the `mysql` or `mysqli` extensions shipped with the PHP sources. See Preface.

Chapter 6. Common Problems with MySQL and PHP

- `Error: Maximum Execution Time Exceeded`: This is a PHP limit; go into the `php.ini` file and set the maximum execution time up from 30 seconds to something higher, as needed. It is also not a bad idea to double the RAM allowed per script to 16MB instead of 8MB.
- `Fatal error: Call to unsupported or undefined function mysql_connect() in ...`: This means that your PHP version isn't compiled with MySQL support. You can either compile a dynamic MySQL module and load it into PHP or recompile PHP with built-in MySQL support. This process is described in detail in the PHP manual.
- `Error: Undefined reference to 'uncompress'`: This means that the client library is compiled with support for a compressed client/server protocol. The fix is to add `-lz` last when linking with `-lmysqlclient`.
- `Error: Client does not support authentication protocol`: This is most often encountered when trying to use the older `mysql` extension with MySQL 4.1.1 and later. Possible solutions are: downgrade to MySQL 4.0; switch to PHP 5 and the newer `mysqli` extension; or configure the MySQL server with `--old-passwords`. (See `Client does not support authentication protocol`, for more information.)

Those with PHP4 legacy code can make use of a compatibility layer for the old and new MySQL libraries, such as this one: http://www.coggeshall.org/oss/mysql2i.

Chapter 7. Enabling Both `mysql` and `mysqli` in PHP

If you're experiencing problems with enabling both the `mysql` and the `mysqli` extension when building PHP on Linux yourself, you should try the following procedure.

1. Configure PHP like this:

   ```
   ./configure --with-mysqli=/usr/bin/mysql_config --with-mysql=/usr
   ```

2. Edit the `Makefile` and search for a line that starts with `EXTRA_LIBS`. It might look like this (all on one line):

   ```
   EXTRA_LIBS = -lcrypt -lcrypt -lmysqlclient -lz -lresolv -lm -ldl -lnsl
   -lxml2 -lz -lm -lxml2 -lz -lm -lmysqlclient -lz -lcrypt -lnsl -lm
   -lxml2 -lz -lm -lcrypt -lxml2 -lz -lm -lcrypt
   ```

   Remove all duplicates, so that the line looks like this (all on one line):

   ```
   EXTRA_LIBS = -lcrypt -lcrypt -lmysqlclient -lz -lresolv -lm -ldl -lnsl
   -lxml2
   ```

3. Build and install PHP:

   ```
   make
   make install
   ```