23

PHP Reference

PHP provides a wide range of functions that are useful when creating database-driven applications. So many, in fact, that it would be unwieldy to list all of them in a book about MySQL. Therefore this reference chapter concentrates on the functions that PHP provides to interface directly with MySQL. This includes the new PHP database abstraction layer which promises to someday unify the various PHP database APIs into a single set of functions.

mysql_affected_rows

- Returns the number of rows affected by the last non-SELECT statement

$num_rows = mysql_affected_rows([$mysql])

mysql_affected_rows returns the number of rows altered in any way by the last statement. Since this only reports on rows that have been changed in some way, it has no meaning when used after a SELECT statement. Also there are a couple of cases where MySQL performs optimizations that affect the result of this function:

UPDATE - It is import to note, as mentioned above, this function returns the number of rows that are changed in some way by the query. This means that an UPDATE query that matches a row, but does not change its value, is not counted. For example, the query UPDATE mytable SET column = 'fnord' will return 0 if every row in the table already has 'fnord' for the value of 'column'.

DELETE - MySQL performs an optimization with deleting the entire contents of a table that makes it impossible to tell the number of rows that were in that table. Therefore, if you delete all of a table using 'DELETE from tablename' with no WHERE clause, this function will return 0.

A specific connection can be specified by passing the connection identifier variable as a parameter to this function. Otherwise, the most recently opened connection is used.

This function should be called immediately after the query you are interested in. This holds true even when using tables that use transactions; this function should be called after the query, not the commit.

Example

```
<? mysql_query("DELETE from people where firstname like 'P%'"); ?>
You have deleted <?= mysql_affected_rows() ?> people.

<? // $mysql is a seperate server connection that was established earlier.
    mysql_query("UPDATE people SET lastname = 'Smith' where
UPPER(lastname)='SMITH'",
         $mysql);  ?>
You have corrected the case in <?= mysql_affected_rows($mysql) ?> instances of
'Smith'.
<? // Note that this will accurately report the number of 'Smith's that were
changed. Any records
     // that were already 'Smith' were not counted because they were not changed,
even though
     // they matched the WHERE clause. ?>
```

mysql_change_user

- Changes the currently authenticated user for a MySQL session

$success = mysql_change_user($username, $password [, $database [, $mysql]])

mysql_change_user re-authenticates a MySQL server connection with the given username and password. If a third argument is give, it is used as the default database if the re-authentication is successful. By default, this function uses the most recently opened MySQL connection. A specific connection can be specified as the forth argument.

This function returns a true value on success and a false value if the re-authentication fails. In the case of failure the authentication information in effect before the function was called stays active (and the default database does not change).

Example

```
<? // Switch users to 'newuser', 'newpass'
     mysql_change_user( 'newuser', 'newpass' );

     // Switch users to 'newuser', 'newpass' and change the default database to
'newdb'
     // If the change is unsuccessful, print a warning.
    if (! mysql_change_user('newuser', 'newpass', 'newdb') { ?>
Warning! Database re-authentication failed!.
    <? }

    // Switch users to 'newuser', 'newpass', using the existing MySQL connection
$mysql.
    mysql_change_user('newuser', 'newpass', '', $mysql); ?>
```

mysql_close

- Closes MySQL connection

$success = mysql_close([$mysql])

mysql_close closes the most recently opened MySQL server connection. A specific connection can be specified as the first parameter. The function returns true if the connection was successfully closed and false if there was an error.

It is generally not necessary to use this function as non-persistent connections are automatically closed at the end of the script where they are used. This function only has effect on non-persistent connections. Persistent connections, opened with *mysql_pconnect*, will be unaffected.

Example

```
<? // Close the most recent connection.
     mysql_close();

     // Attempt to close a persistent connection referenced by the variable $mysql.
     if (! mysql_close( $mysql ) ) ?>
The connection didn't close! This is probably because it was a persistent
connection.
     <? } ?>
```

mysql_connect

- Open a connection to a MySQL Server

$mysql = mysql_connect([$host[, $user [, $password]]])

mysql_connect attempts to open a connection with a MySQL server. If successful this function returns a MySQL connection variable that can be used with most of the MySQL functions to specify this connection. In the case of failure, a false value is returned.

If no arguments are given, PHP attempts to connect to the MySQL server on the local host at port 3306 using the username of the user that owns the PHP process and a blank password. The hostname can be specified as the first parameter. If a TCP connection is desired on any port other than 3306, it is specified as part of the hostname, using a ':' separator. If a local Unix socket connection is needed, the pathname of the socket should be specified in the same manner. The second argument specifies the username, and the third specifies the password.

Note: If PHP is running in "safe mode," only the default hostname, port, username and password are allowed.

Note: Care should be taken when specifying password information within a PHP script. Other users of the same machine will likely be able to view the script and see the password. Visitors over the web will not be able to view the script, however, so this is only an issue if you share the server machine with other people.

If more than one call is made to *mysql_connect* using identical arguments, all of the calls after the first will return the connection variable created with the first call (if that connection is still open).

Example

```
<? // Connect to the mysql server on the local host at port 3306 using the username
of the
     // owner of the PHP process and a blank password:
    $mysql = mysql_connect();
    // Connect to the mysql server on the localhost using the Unix socket
/tmp/mysql.sock and
    // the default username and password
    $mysql = mysql_connect( 'localhost:/tmp/mysql.sock' );

    // Connect to the mysql server at my.server.com, port 3333 using the username
'me' and a
    // blank password
    $mysql = mysql_connect('my.server.com:3333', 'me');

   // Connect to the mysql server on the localhost, port 3306, with username 'me'
and the
   // password 'mypass'
   $mysql = mysql_connect('', 'me', 'mypass');

   // Use the same connection parameters as above:
   $mysql2 = mysql_connect('', 'me', 'mypass');
   // $mysql2 is not a new connection, but rather another reference to the same
connection
   // as $mysql
?>
```

mysql_create_db

- creates a new database

$success = mysql_create_db ($database [, $mysql]

mysql_create_db attempts to create a new database using the given database name. The database is created using the most recently mysql connection. A specific connection can be specified as the second argument.

Note: *mysql_createdb* is an alias to *mysql_create_db* provided for backwards compatibility. It should not be used in new scripts.

The function returns true in the case of success and false on failure.

Example

```
<? // Create a new database called 'newdb'
     mysql_createdb("newdb");

     // Attempt to create a database called 'newdb2' using the connection given by
the
     // $mysql variable

     if (! mysql_createdb("newdb2", $mysql) ) { ?>
Attempt to create database newdb2 failed!
      <? } ?>
```

mysql_data_seek

- Move internal result pointer

$success = mysql_data_seek ($result, $row_number)

mysql_data_seek moves the internal pointer within the given result set variable to a specific row. The next attempt to read a row from the result set (such as via *mysql_fetch_row*) will return the row specified here. The first row of a result set is always 0.

This function returns true on success and a false value in the case of failure. This function will fail if the given row number does not exist in the result set. This commonly occurs when using this function to move back to the beginning of a result set by seeking to row 0. This will always work, unless the result set is empty (i.e. the query did not return any rows), in which case there is no row 0 and this function will fail. Therefore, it may be wise to check the number of rows in the result set, using mysql_num_rows to make sure the row you are seeking to exists.

Example

```
<? // Reset the result pointer (given by the variable $result) to the third row of
the result set
      mysql_data_seek( $result, 2 );

     // Reset the result pointer back to the beginning of the result set only if
this is possible.
     if (mysql_num_rows( $result ) > 0 ) {
            mysql_data_seek( $result, 0 );
     }
?>
```

mysql_db_name

- Read a database name from a result set

$dbname = mysql_db_name ($result, $row_num [, $unused])

mysql_db_name returns the name of a database from a result variable created from a call to *mysql_list_dbs*. The second argument to this function indicates which database name out of all of those in the result set to return. The first database in the result set is always 0. The number of database names in the result set can be obtained from *mysql_num_rows*.

This function returns a false value in the case of an error. Supplying a row number that does not exist in the result set will result in an error.

> This function is implemented as an alias to mysql_result. Because of this, it is possible to supply a third argument to this function, which represents the name of a field in the result set. However, this function is not useful in the context of mysql_db_name.
>
> mysql_dbname is available as an alias to this function for backwards compatibility but should not be used for new scripts.

Example

```
<? // $result is the result of a call to mysql_list_dbs. It contains the names of
all of the databases
    // available to the user.
    for ( $i = 0; $i < mysql_num_rows( $result ); $i++ ) {
           echo mysql_db_name( $result, $i );
    }
?>
```

mysql_db_query

- Executes a query with a specific default database

$result = mysql_db_query($database, $query [, $mysql])

mysql_db_query executes a SQL query given as the second argument, using the first argument as the default database for the query. The query is executed using the most recently opened database connection. A specific server connection can be specified using a third argument. The database specified as the first argument becomes the new default database for the connection.

If the query is a SELECT query and is successful, the function returns a result set variable that can be used with functions like *mysql_fetch_row* to retrieves the contents of the results. If the query is a non-SELECT query (such as INSERT, UPDATE or DELETE) and is successful, the function returns a true value. If the query fails, a false value is returned.

The function *mysql* is provided as an alias for backwards compatibility, but should not be used with new scripts.

Example

```
<? // Select all of the rows from the 'people' table in the 'mydb' database
     $result = mysql_db_query( "mydb", "select * from people" );
     // $result now contains a result set that can be used with myql_fetch_row() to
read the values.

     // Perform a query against the 'mydb' database using the connection specified
with the
     // $mysql connection variable and check to make sure it's a valid result set.
     if (! $result = mysql_db_query( "mydb",
                     "select firstname from people where firstname like 'P%'",
$mysql ) ) { ?>
               The query was unsuccessful!
       <? } ?>

     // Insert a new row into the table 'people' in the database 'mydb' and check
to make sure
     // the insert worked.
     if (! $result = mysql_db_query("mydb",
                     "insert into people values ('John', 'Doe')" ) { ?>
              The insert failed!
       <? } ?>
```

mysql_drop_db

- Deletes a database

$success = mysql_drop_db($database [, $mysql])

mysql_drop_db attempts to delete the given database. This is an irrevocable operation which will permanently delete all of the data within the database. This function uses the most recently opened server connection. A specific server connection can be specified with the second argument.

The function returns true if the database is successfully dropped and false in the case of an error.

The function mysql_dropdb is provided as an alias for backwards compatability but should not be used with new scripts.

Example

```
<? // Drop the database 'olddb'
     if (! mysql_drop_db( "olddb" ) ) { ?>
Error deleting the database!
      <? }

     // Drop the database 'otherdb' using the connection given by the $mysql
variable
    mysql_drop_db( "otherdb", $mysql );
?>
```

mysql_errno

- Return the last error code

$error_code = mysql_errno([$mysql])

mysql_errno returns the MySQL-specific error code for the last MySQL error that occurred during the current connection. Any successful MySQL-related function call resets the value of this function to 0. Because of this, you should always call this function immediately after the function you are interested in checking for errors.

Example

```
The MySQL-related function returned an error-code of <?= mysql_errno() ?>.
```

mysql_error

- Returns the text description of the last error

$error = mysql_error([$mysql])

mysql_error returns the human-readable description of the last MySQL error that occurred during the current connection. Any successful MySQL-related function call resets the value of this function to an empty string. Because of this, you should always call this function immediately after the function you are interested in checking for errors.

Example

```
<? // The variable $mysql is a MySQL connection variable
The last MySQL-related error was <?= mysql_error($mysql) ?>.
```

mysql_escape_string

- Escapes a string for use in a mysql_query.

$escaped_string = mysql_escape_string ($string)

mysql_escape_string takes a string as an argument and returns a copy of that string that has any special characters escaped so that is is safe to use in a MySQL-query. Specifically, it escapes any single quotes "'" as a double-single quote ("''").

Example

```
<? // $value contains some data, which may contain special characters.
$query = "INSERT into table values ('" + mysql_escape_string($value) + "'";
// $query now contains a SQL query that is safe to execute. ?>
```

mysql_fetch_array

- Retrieve a row of a result set as an array

$row_values = mysql_fetch_array ($result [, $type_of_array])

mysql_fetch_array returns an array of values from the result set pointed to by the first argument. The function returns all of the fields in the next row of data in the result set. It also advances the internal "pointer" of the result set, so that the next call to *mysql_fetch_array* (or any similar function) will return the next row in the result set.

By default, the array returned by this function is both a numerically indexed array and an associative array. The fields in the query are stored using numeric indices with 0 being the first field in the SQL statement. The fields are also stored as an associative array with the names of the fields being the keys. If you want to use just one type of array, passing a second argument to the function can set that behavior. If the argument is MYSQL_NUM a numerically indexed array is used; if the argument is MYSQL_ASSOC an associative array is used and if the argument is MYSQL_BOTH both types of arrays are used (this is the default).

When using an associative array some care should be taken to make sure the names of the fields are unique. If two or more fields in the query have the same name, only the last field is available via the associative array. The other fields must be accessed via their numeric index.

Prior to PHP 4.05, a field that has a null value within a row would not show up within the associative array. This could create problems when checking the array for field names that should exist. As of PHP version 4.05, this problem has been fixed.

The function returns a false value if there are no more rows of data in the result set.

Example

```
<? // $result is a result set variable from the query "SELECT firstname, lastname
from People"
$firstrow = mysql_fetch_array( $result );
?> The first person in the result set is <?= $firstrow[0] ?> <?= $firstrow[1]
?>.<br>
$secondrow = mysql_fetch_array( $result  ); ?>
The second person in the result set is
    <?= $secondrow['firstname'] ?> <?= secondrow['lasrtname''] ?>
// Fetch the third row as only associatve
$thirdrow = mysql_fetch_array( $result, MYSQL_ASSOC ); ?>
The third row of data has <?= $thirdrow['firstname'] ?> <?= $thirdrow['lastname']
?>
```

mysql_fetch_assoc

- Retrieve a row of a result set as an associative array

$assoc_array = mysql_fetch_assoc ($result)

mysql_fetch_assoc returns an associative array of values from the result set pointed to by the first argument, with the names of the fields in the SQL query as the keys of the array.

The function returns all of the fields in the next row of data in the result set. It also advances the internal "pointer" of the result set, so that the next call to *mysql_fetch_assoc* (or any similar function) will return the next row in the result set.

Some care should be taken to make sure the names of the fields are unique. If two or more fields in the query have the same name, only the last field is available via the associative array.

The function returns a false value if there are no more rows of data in the result set.

Example

```
<? // $result is a result variable from the query "SELECT * from People"
     $firstrow = mysql_fetch_assoc( $result ) ?>
The first row of data has is <?= $firstrow['firstname'] ?> <?=
$firstrow['lastname'] ?>
```

mysql_fetch_field

- Retrieve meta-information about a field from a result set

$field_info = mysql_fetch_field($result [, $field_number])

mysql_fetch_fields returns an object containing information about a field contained in a result set. The first argument is a result set variable and the second indicates which field in the result set to retrieve information about. If no second argument is provided, the first field that has not yet been retrieved is used. Thus, successive calls to *mysql_fetch_fields* can be used to retrieve information about all of the fields in a result set.

The object returned by the function contains the following properties:

name - The name of the field as specified in the SQL query
table - The name of the table to which the field belongs, if any
max_length - The maximum length of the column.
not_null - Indicates if the field can be set to a null value (true if it cannot)
primary_key - Inticates if the field is part of a primary key (true if it is)
unique_key - Indicates if the field is part of a unique key (true if it is)
multiple_key - Indicates if the field is part of a non-unique key (true if it is)
numeric - Indicates if the field is a numeric type (true if it is)
blob - Indicates if the field is a BLOB type (true if it is)
type - The type of the field
unsigned - Indicates if the field is an unsigned numeric (true if it is)
zerofill - Indicates if the field is automatically filled with null-characters (0's)

Example

```
<? // $result is a result set created from the query "select firstname, lastname
from People"
      $fname_info = mysql_fetch_field( $result, 0 );
?>
Info about the '<?= $fname_info->table ?>.<?= $fname_info->name ?>' field:<br>
Maximum length: <?= $fname_info->max_length<br>
Type: <?= $fname_info->type ?><br>

<? $lname_info = mysql_fetch_field( $result ); // Returns the next field that
hasn't been
```

```
                                                            // returned yet;
which is 'lastname' in this case.
       if ( $lname_info->multiple_key ) {
?>
<?= $lname_info->table ?>.<?= $lname_info->name ?> is an indexed field.
<? } ?>
```

mysql_fetch_lengths

- Retrieve the field-lengths for the last row in a result set

$lengths = mysql_fetch_lengths($result)

mysql_fetch_lengths returns an array of field lengths for the last row of data returned from a result set. The first argument to the function is a result set variable pointing to a result set that has had at least one row read from it. The values of this array correspond to the values of the fields as returned by *mysql_fetch_row* or a similar function. The values are the actual length of data that was returned.

The function returns a false value in the case of an error.

Example

```
<? // $result is a result set variable created earlier. It must be accessed at
least once using a
     // function such as mysql_fetch_row
     $lengths = mysql_fetch_lengths( $result );
?>
The first field in the most recent row fetched was <?= $lenghts[0] ?> characters
long.
```

mysql_fetch_object

- Retrieve a row of a result set as an object

$object = mysql_fetch_object($result [, $data_type])

mysql_fetch_object returns an object created from the result set pointed to by the first argument. This contains all of the fields in the next row of data in the result set. It also advances the internal "pointer" of the result set, so that the next call to *mysql_fetch_object* (or any similar function) will return the next row in the result set.

By default, the object's fields are the names of the fields in the query and also the numeric indices of the query. If you want the fields to have just the names or just the numeric indices, that behavior can be set by passing a second argument to the function. If the argument is MYSQL_NUM the numerically indexed fields are used; if the argument is MYSQL_ASSOC the field names are used and if the argument is MYSQL_BOTH both sets of fields are used (this is the default).

The function returns a false value if there are no more rows of data in the result set.

Example

```
<? // $result is a result set variable from the query "SELECT firstname, lastname
from People"
$firstrow = mysql_fetch_object( $result );
?> The first person in the result set is <?= $firstrow->0 ?> <?= $firstrow->1
?>.<br>
```

```
$secondrow = mysql_fetch_object( $result  ); ?>
The second person in the result set is
    <?= $secondrow->firstname ?> <?= secondrow->lasrtname  ?>
// Fetch the third row as only associatve
$thirdrow = mysql_fetch_array( $result, MYSQL_ASSOC ); ?>
The third row of data has <?= $thirdrow->firstname ?> <?= $thirdrow->lastname ?>
```

mysql_fetch_row

- Retrieve a row of a result set as an array

$array = mysql_fetch_row($result)

mysql_fetch_row returns an numerically-indexed array of values from the result set pointed to by the first argument, with the indexes in the same order as in the SQL query that created the result set. The function returns all of the fields in the next row of data in the result set. It also advances the internal "pointer" of the result set, so that the next call to *mysql_fetch_row* (or any similar function) will return the next row in the result set.

Unlike in the associative array functions, like *mysql_fetch_assoc*, the array returned by this function will contain all of the fields of the result set, even if some fields had the same name (or no name at all). This function is also slightly faster than the other fetch functions, but it is a minimal difference and the other functions should be used when appropriate.

The function returns a false value if there are no more rows of data in the result set.

Example

```
<? // $result is a result variable from the query "SELECT count(*) from People"
     $firstrow = mysql_fetch_row( $result ) ?>
There are <?= $firstrow[0] ?> people in the People table.
```

mysql_field_flags

- Retrieve the flags associated with a field in a result set

$flags = mysql_field_flags($result, $which_field)

mysql_field_flags returns a string containing any special flags associated with a field in the result set. The first argument is the result set variable and the second argument is the number of the field as given in the SQL query which created the result set. The first field is 0.

The flags are returned as a string with the flag names separated by a space. The following flags are currently possible:

not_null - The field cannot contain a NULL value
primary_key - The field is a primary key
unique_key - The field is a unieque key (there can be no duplicates)
multiple_key - The field is a multiple key (they can be duplicates)
blob - The field is a BLOB type, such as BLOB or TEXT
unsigned - The field is an unsigned numeric-type
zerofill - The field will fill any unused space with zero's up to the maximum field length
binary - The field may contain binary data and will use binary-safe comparisons

enum - The field is an enumeration, which can contain one of a number of pre-defined values
auto_increment - The field is an auto-increment field
timestamp - The field is an automatic timestamp field

> The function mysql_fieldflags is provided as an alias to this function for backwards compatability. It should not be used for new scripts.

Example

```
<? // $result is a result set variable created by the SQL query "Select firstname
from People"
     $flags = mysql_field_flags( $result, 0 );
?>
The first 'firstname' field in the People table has these flags: <?= $flags ?>
```

mysql_field_name

- Retrieves the name of a field in a result set

$field_name = mysql_field_name($result, $which_field)

mysql_field_name returns the name of a field within a result set given by the first argument. The second argument is the number of the field within the SQL query that created the result set. The first field of the query is always 0.

Example

```
<? // $result is a result set variable created by a SQL query
     $field_name = mysql_field_name( $result, 0 );
?>
The first field of the SQL query was '<?= $field_name ?>'
```

mysql_field_len

- Returns the length of a field within a result set

$length = mysql_field($result, $which_field)

mysql_field_len returns the maximum length of a field within a result set given by the first argument. The second argument is the number of the field within the SQL query that created the result set. The first field of the query is always 0.

This function does not return the length of the data within a result set, but the length of the field as defined in the database table.

> The function *mysql_fieldlen* is provided as an alias for backwards compatibility. It should not be used in new scripts.

Example

```
<? // $result is a result set variable created by a SQL query
      $len = mysql_field_name( $result, 0 );
?>
The first field of the SQL query has a length of <?= $len ?>
```

mysql_field_seek

- Set the internal field pointer in a result set to a specific field

$success = mysql_field_seek($result, $which_field)

mysql_field_seek sets the internal field pointer within a result set given by the first argument. The second argument is the number of the field within the SQL query that created the result set. The first field of the query is always 0.

Once the field pointer has been set using this function, the next call to *mysql_fetch_field* will return this field.

The function return a true value of the field pointer is successfully set and a false value in the case of an error.

Example

```
<? // $ result is a result set variable created by a SQL query
         mysql_field_seek( $result, 0 );
         // The next call to mysql_fetch_field will return the first field in the
result set.
?>
```

mysql_field_table

- Retrieve the table name for a field within a result set

$table_name = mysql_field_table($result, $which_field)

mysql_field_table returns the name of the table that contains the given field within the result set given by the first argument. The second argument is the number of the field within the SQL query that created the result set. The first field of the query is always 0.

Example

```
<? // $result is a result set variable created by a SQL query
     $table = mysql_field_table( $result, 0 );
?>
The first field of the query belongs to table <?= $table ?>
```

mysql_field_type

- Retrieve the type of a field in a result set

$type = mysql_field_type($result, $which_field)

mysql_field_type returns the PHP type of a field within the result given by the first argument. The second argument is the number of the field within the SQL query that created the result set. The first field of the query is always 0.

The type of the field is returned as a string, such as ‘int’, ‘string’, ‘real’ and others.

Example

```
<? // $result is a result set variable created by a SQL query
     $type = mysql_field_type( $result, 0);
?>
The first field of the query is a <?= $type ?>
```

mysql_free_result

- Frees the memory used by a result set

$success = mysql_free_result($result)

mysql_free_result frees any memory used by a result set created by a SQL query. This is done automatically at the end of any script that uses a result set, therefore it is never necessary to use this function. However, when dealing with large result sets it may be beneficial to call this function as soon as you are done with the result set, in order to free of the memory used.

The function returns true on success and false in the event of an error.

> The function *mysql_freeresult* is an alias to this function provided for backwards compatibility. It should not be used in new scripts.

Example

```
<? // $result is the result set created by a SQL query that returned a large number
of rows
     mysql_free_result( $result );
     // The memory used by $result has been freed and $result can no longer be used
as a result set.
?>
```

mysql_insert_id

- Get the id generated from the previous INSERT operation

$id = mysql_insert_id([$mysql])

mysql_insert_id returns the last id that was automatically generated from an auto_increment column for a certain connection. Each connection remembers it's own most recently generated id, so this function works on a per-connection bases. By default it uses the most recently opened MySQL connection. A specific can be specified as the sole parameter to this function.

The function returns a false value if there have been no auto-generated ids for the given connection.

> This function performs the same operation as the MySQL-specific LAST_INSERT_ID() SQL statement.

Example

```
<? // An INSERT statement has been executed on an auto-increment table
     $id = mysql_insert_id();
?> The ID of the row that was just created is <?= $id ?>
```

mysql_list_dbs

- Retrieve a list of databases on a MySQL server

$result = mysql_list_dbs([$mysql])

mysql_list_dbs retrieves a list of databases that are available on a MySQL server. By default, the most currently opened server connection is used. A specific server connection can be specified as the sole parameter.

This function returns a result set variable that contains the names of the databases. The data within the result set can be viewed using the *mysql_db_name* function.

> The function *mysql_listdbs* is provides as an alias to this function for backwards compatability. It should not be used for new scripts.

Example

```
<? $result = mysql_list_dbs();
     // $result is a now a result set variable containing a list of the databases
on the server
     // most recently opened. mysql_db_name() can be used to retrieve the actual
names

     // $mysql is a connection variable that was created earlier in the script
     $another_result = mysql_list_dbs( $mysql );
      // $another_result is a reset set variable containing a list of the database
on the server used by
      // the $mysql connection.
?>
```

mysql_list_fields

- Retrieve a list of fields for a database table

$result = mysql_list_fields($database, $table[, $mysql])

mysql_list_fields retrieves a list of fields for the given table within the given database. By default it uses the most recently opened server connection. A specific server connection can by given as the third parameter.

The function returns a result set variable that contains information about the fields. This information can be retrieved through functions like *mysql_field_name*, *mysql_field_type*, *mysql_field_len* and *mysql_field_flags*.

> The function *mysql_listfields* is provided as an alias to this function for backwards compatibility. It should not be used in new scripts.

Example

```
<? // Find out about the fields in the 'People' table of 'mydb'
      $result = mysql_list_fields( 'mydb', 'People' );
     // $result is a result set variable that can be used with mysql_field_name and
other similar
     // functions to examine the fields in this result set.
?>
```

mysql_list_tables

- Retrieve a list of tables in a MySQL database

$result = mysql_list_tables($database[, $mysql])

mysql_list_tables retrieves a list of tables for the given database. By default it uses the most recently opened server connection. A specific server connection can by given as the second parameter.

The function returns a result set variable that contains the name of the tables in the database. These names can be retrieved through the *mysql_tablename* function.

The function *mysql_listtables* is provided as an alias to this function for backwards compatibility. It should not be used in new scripts.

Example

```
<? // Find what tables are available in the 'mydb' database...
     $result = mysql_list_tables( 'mydb' );
    // $result is a result set variable that can be used with mysql_tablename to
get the names
?>
```

mysql_num_fields

- Find the number of fields in a result set

$num_fields = mysql_num_fields($result)

mysql_num_fields returns the number of fields contained in each row of the given result set. The result set variable is one that was returned from a function such as *mysql_query*, or any other function that returns a result set.

Note: The function *mysql_numfields* is provided as an alias for backwards compatibility. It should not be used in new scripts.

Example

```
<? // $result is a result set variable returned from a SQL query ?>
The query contains <?= mysql_num_fields( $result ) ?> fields.
```

mysql_num_rows

- Find the number of rows in a result set

$num_rows = mysql_num_rows($result)

mysql_num_rows returns the number of rows contained in the given result set. The result set variable is one that was returned from a function such as *mysql_query*, or any other function that returns a result set.

Example

```
<? // Result is a result set variable that was created from a SELECT SQL query ?>
The query returned <?= mysql_num_rows( $result ) ?> rows...
```

mysql_pconnect

- Open a persistent connection to a MySQL Server

$mysql = mysql_pconnect([$host[, $user [, $password]]])

mysql_pconnect attempts to create (or used an existing) persistent connection with a MySQL server. If successful this function returns a MySQL connection variable that can be used with most of the MySQL functions to specify this connection. In the case of failure, a false value is returned.

Before creating a new connection, this function looks for an existing persistent connection that was created with the same arguments. If one is found, it is used instead. Persistent connections stay available even between PHP scripts, as long as everything is running within the same PHP process.

That is, if PHP is running within a web server (such as via Apache's mod_php), persistent connections stay available to every page (see Chapter XX: PHP for a few caveats). Unless *mysql_connect*, when the script ends, the connection stays alive and can be used by another PHP script. On the other hand, if PHP is being run as a CGI, a new process is created every time and therefore the persistent connections are closed at the end of each script and this function is identical to *mysql_connect*.

If no arguments are given, PHP attempts to connect to the MySQL server on the local host at port 3306 using the username of the user that owns the PHP process and a blank password. The hostname can be specified as the first parameter. If a TCP connection is desired on any port other than 3306, it is specified as part of the hostname, using a ':' separator. If a local Unix socket connection is needed, the pathname of the socket should be specified in the same manner. The second argument specifies the username, and the third specifies the password.

> If PHP is running in "safe mode," only the default hostname, port, username and password are allowed.
>
> Care should be taken when specifying password information within a PHP script. Other users of the same machine will likely be able to view the script and see the password. Visitors over the web will not be able to view the script, however, so this is only an issue if you share the server machine with other people.

Example

```
<? // Get a persistent connection to the mysql server on the local host at port
3306 using the
     // username of the owner of the PHP process and a blank password:
    $mysql = mysql_pconnect();

    // Get a persistent connection to the mysql server on the localhost using the
Unix socket
    // /tmp/mysql.sock and the default username and password
    $mysql = mysql_pconnect( 'localhost:/tmp/mysql.sock' );

    // Get a persistent connection to the mysql server at my.server.com, port 3333
using the
    // username 'me' and a blank password
    $mysql = mysql_pconnect('my.server.com:3333', 'me');

   // Get a persistent connection to the mysql server on the localhost, port 3306,
with username
   // 'me' and the password 'mypass'
   $mysql = mysql_pconnect('', 'me', 'mypass');
?>
```

mysql_query

- Executes a SQL query

$result = mysql_query($query[, $mysql])

mysql_query executes the given SQL query. The query is executed using the most recently opened database connection. A specific server connection can be specified as the second argument.

If the query is a SELECT query and is successful, the function returns a result set variable that can be used with functions like *mysql_fetch_row* to retrieves the contents of the results. If the query is a non-SELECT query (such as INSERT, UPDATE or DELETE) and is successful, the function returns a true value. If the query fails, a false value is returned.

Example

```
<? // Select all of the rows from the 'people' table
     $result = mysql_query( "select * from people" );
     // $result now contains a result set that can be used with myql_fetch_row() to
read the values.

     // Perform a query using the connection specified with the $mysql connection
variable
     // and check to make sure it's a valid result set.
     if (! $result = mysql_query(
               "select firstname from people where firstname like 'P%'", $mysql ) )
{ ?>
               The query was unsuccessful!
       <? } ?>

     // Insert a new row into the table 'people' and check to make sure
     // the insert worked.
     if (! $result = mysql_db_query("insert into people values ('John', 'Doe')" ) {
?>
              The insert failed!
       <? } ?>
```

mysql_unbuffered_query

- Execute a SQL query, without immediately fetching the result rows

$result = mysql_unbuffered_query($query[, $mysql])

mysql_unbuffered_query executes the given SQL query without immediately storing the result. The query is executed using the most recently opened database connection. A specific server connection can be specified as the second argument.

If the query is a SELECT query and is successful, the function returns a result set variable that can be used with functions like *mysql_fetch_row* to retrieves the contents of the results. Unlike *mysql_query*, the results are fetched as requested one row at a time. For large queries this provides faster performance and lower memory usage. However, this also means that the *mysql_num_rows* function cannot be used with result sets created with *mysql_unbuffered_query* because there is no way to know how many rows are in the result set.

Because of the internal MySQL workings of this function, it is important that you fetch all of the rows of a result set created with *mysql_unbuffered_query* before creating a new query. Failure to do so may create unpredictable results.

Example

```
<? // Select all of the rows from the 'people' table
     $result = mysql_ubuffered_query( "select * from people" );
     // $result now contains a result set that can be used with myql_fetch_row() to
read the values.
     // Because this is an unbuffered query, no system resources are being used for
PHP right now
     // for this result set. The rows will be fetched from the MySQL server as you
request them
     // But remember that you have to request them all before creating a new
query...
```

mysql_result

- Retrieve a single field of data from a result set

$field = mysql_result($result, $row[, $column])

mysql_result retrieves a single field, or "cell", of information from the given result set (which was created from a call to *mysql_query* or some similar function that returns a result set). The row that contains the field is given as the second parameter. The first row of the result set is always 0.

By default, the first field of the row is returned. A specific field can be specified as a third parameter. This field can be specified by using the fields position (the first field is always 0), name as given in the SQL query, or the qualified SQL name of the field (that is the name of the table and the name of the field separated by a dot).

mysql_result deals with the position inside of a result set differently than functions that return an entire row, such as *mysql_fetch_row*. Therefore you should not mix *mysql_result* with any other function that reads data from a result set.

Example

```
<? // $result is a result set created by the SQL query:
     //  SELECT firstName as fname, lastName from People
The first first name in the 'People' table is <?= mysql_result( $result, 0 ) ?><br>
The first last name in the 'People' table is <?= mysql_result( $result, 0 , 1 )
?><br>
The second first name in the 'People' table is <?= mysql_result( $result, 1,
'fname' ) ?><br>
The second last name in the 'People' table is <?= mysql_result( $result, 1,
'lastName' ) ?><br>
The third last name in the 'People' table is <?= mysql_result( $result, 2,
'People.lastName' )?><br>
// Since the name of 'firstName' is specified as the alias 'fname' we cannot use
the
// table.field notation, but must use the alias instead
```

mysql_select_db

- Select a new default database

$success = mysql_select_db($database[, $mysql])

mysql_select_db selects a new default database. Any SQL query that uses tables within specifying a database will use this database.

By default, this function sets the database on the most recently opened connection. A specific connection can be given as the second argument. If no specific connection is given, and no current connection exists, PHP will attempt to open a new connection as if *mysql_connect* was called with no parameters.

> The function *mysql_selectdb* is provided as an alias to this function for backwards compatibility. It should not be used in new scripts.

This function returns true on success and false in the case of failure.

Example

```
<? mysql_select_db( 'mydb');
     // Now the query 'SELECT * from People' will use the People table in 'mydb'
     mysql_selectdb('myotherdb');
     // Now the query 'SELECT * from People' will use the People table in
'myotherdb'
?>
```

mysql_tablename

- Get table name of field

$table = mysql_tablename($result, $which_table[, $unused])

mysql_tablename returns the name of a table from a result variable created from a call to *mysql_list_tables*. The second argument to this function indicates which table name out of all of those in the result set to return. The first table in the result set is always 0. The number of table names in the result set can be obtained from *mysql_num_rows*.

This function returns a false value in the case of an error. Supplying a row number that does not exist in the result set will result in an error.

> This function is implemented as an alias to *mysql_result*. Because of this, it is possible to supply a third argument to this function, which represents the name of a field in the result set. However, this function is not useful in the context of *mysql_tablename*.

Example

```
<? // $result is the result of a call to mysql_list_tables contains the names of
all of the tables
    // available to the user.
    for ( $i = 0; $i < mysql_num_rows( $result ); $i++ ) {
           echo mysql_tablename( $result, $i );
    }
```

```
?>
```

mysql_get_client_info

- Retrieve information about the MySQL client library

$info = mysql_get_client_info()

mysql_get_cliet_info returns a string of information about the MySQL client library that PHP is using. Currently it returns the version number of the library.

Example

```
PHP is using MySQL client library version <?= mysql_get_client_info() ?>.
```

mysql_get_host_info

- Retrieve information about a MySQL server connection

$info = mysql_get_host_info([$mysql])

mysql_get_host_info returns a string of information about a MySQL server connection. The information indicates the type of connection (TCP or Unix socket) and the port used. By default this function returns information about the most recently opened server connection. A specific connection can be given as the sole parameter.

Example

```
You are currently connect to MySQL on <?= mysql_get_host_info() ?>
```

mysql_get_proto_info

- Retrieves information about the protocol used in a MySQL connection

$info = mysql_get_proto_info([$mysql])

mysql_get_proto_info returns a string of information about the protocol used in a MySQL server connection. This information currently contains the protocol version number. By default this function returns information about the most recently opened server connection. A specific connection can be given as the sole parameter.

Example

```
You are connected to MySQL using protocol <?= mysql_get_proto_info() ?>
```

mysql_get_server_info

- Retrieves information about a MySQL server

$info = mysql_get_server_info([$mysql])

mysql_get_server_info returns a string of information about a MySQL server that the script is current connected to. This information is currently the version of MySQL that the server is running. By default the function uses the most recently opened server connection. A specific connection can be given as the sole parameter.

Example

```
The server you are connected to is using MySQL version <?= mysql_get_server_info()
?>
```

The following functions are part of the experimental database abstraction layer dbx. Care must be taken when mixing these functions with the mysql functions above. See Chapter XX: PHP for information on the things to watch out for while using these functions.

dbx_close

- Close a database connection

dbx_close($db_connection)

dbx_close closes an open database connection. The given connection variable must be an active connection variable created by *dbx_connect*. PHP will automatically close any open connections at the end of the script. Therefore it is usually not necessary to call this function.

The function returns true on success and false in the case of an error.

Example

```
<?   // $dbconn is an active connection variable created with dbx_connect
       dbx_close( $dbconn ); // The connection is now closed
?>
```

dbx_connect

- Connect to a database

$dbconn = dbx_connect ($db_type, $host, $database, $username, $password [, $make_persnistent])

dbx_connect attempts to create a connection to a database. The first argument is the type of database that you are connecting to. For MySQL this will be “mysql”.

The second argument is the hostname of the MySQL server. If a TCP connection is desired on any port other than 3306, it is specified as part of the hostname, using a ‘:’ separator. If a local Unix socket connection is needed, the pathname of the socket should be specified in the same manner (with the hostname given as ‘localhost’). The third argument is the default database to be used for the connection., The fourth argument is the username and the fifth argument is the password.

It is possible to create this connection as a persistent connection by passing the constant DBX_PERSISTENT as a sixth argument. If this is done, PHP will first check to see if there is an existing persistent connection created with the same parameters. If so, that connection will be used instead of creating a new one. In addition, the connection will not be closed automatically at the end of the script but will be kept open to be re-used by another script.

If the connection is successful a dbx connection variable is returned that can be used with the other dbx functions. If there is an error, the function returns false.

Besides being used with the other dbx functions, the connection variable returned by this function is also an object with three available properties. The first property is ‘database’ which is the name of the default database selected in the connection. The second property is 'module' which returns a constant associated with the type of database chosen ("mysql" in our case). The third property is 'handle' which is an active connection variable specific to the type of database chosen. In our case, this is an active MySQL connection variable that can be used with any of the mysql_* functions..

Example

```
<? // Connect to a MySQL server running on my.server.com with username 'me' and
     // password 'mypass' using 'mydb' as the default database.
     $dbconn = dbx_connect( 'mysql', 'my.server.com', 'mydb', 'me', 'mypass' );
     if ( ! $dbconn ) { echo "There has been an error..."; }
     .else { // The connection was successful! You have an active connection
variable that
                // can be used with the other dbx functions. This variable also has
the property
                // $dbconn->handle which can be used with any of the mysql_*
functions.
      }

      // Connect to a MySQL server running on the localhost with the socket
      //  /tmp/mysql.sock using the default database 'mydb', username 'me' and
password 'mypass'
      // Also make this a persistent connection that will remain available to other
scripts once
      // this script is finished.
      $dbconn = dbx_connect('mysql', 'localhost:/tmp/mysql.sock',
                            'mydb', 'me', 'mypass', DBX_PERSISTENT );
?>
```

dbx_error

- Return the last dbx-related error message

$error = dbx_error($dbconn)

dbx_error returns the most recent error involving the database-specific module used by the given dbx connection variable. This is not necessarily that last error that happened in this connection, but the last error that happened in any connection using that database-specific module. Since we are using the MySQL module, this function will return the last error encountered by any MySQL connection.

Example

```
<? // $dbconn is an active dbx connection variable created with dbx_connect ?>
The last MySQL-related error is <?= dbx_error( $dbconn ) ?>
```

dbx_query

- Execute a SQL query

$dbresult = dbx_query($dbconn, $query[, $flags])

dbx_query executes the given SQL query using the given database connection. If the query fails the function returns false. If the query succeeds and has data (e.g., a SELECT query) the function returns a dbx result object (see below). If the query succeeds and has no data (e.g., an INSERT, UPDATE or DELETE query), the function returns a true value.

For queries with data, the returned dbx result object is an object with several properties:

handle
: This is a module-specific result set. Since we are using MySQL, this is a MySQL result set variable that can be used with any of the mysql_* functions above that take a result set.

cols
: This is the number of fields per row in the result set.

rows
: This is the number of rows of data in the result set.

data
: This is a two-dimensional array containing all of the data in the result set. The first dimension of this array is the row of the data. The first row is always 0. The second dimension of the array specified the field you are interested in. By default, this dimension contains both numeric and associative indices. The numeric index is the position of the field as given in the SQL query (the first field is always 0). The associative index is the name of the field as given in the SQL query (fields with no name have no associative index). The numeric and associative indices access the exact same data, so that if you modify a value using the numeric index, the corresponding associative element will change as well.

info
: This is a two-dimensional array which provides access to information about the fields in the query. The first dimension of the array is a string given the type of information request. Currently 'name' and 'type' are the only valid values here. The second dimension is the position of the field desired as given in the SQL query. The first field is always 0.

The default behavior of this function can be modified by providing one or flags as the third argument. Multiple flags can be used by combining them with Boolean-or.

DBX_RESULT_INFO - This flag causes the info property to be available in the result object.
DBX_RESULT_INDEX - The flag causes the second dimension of the data property to contain numeric indices.
DBX_RESULT_ASSOC - This flag causes the second dimension of the data property to contain associative indices. This also causes the info property to be available.

Note that the behavior given by all three flags is the default behavior. Therefore, the only purpose to specify the flags is to *remove* behavior. For example, if you didn't want the info property, you would have to specify the DBX_RESULT_INDEX flag by itself, which would eliminate the DVX_RESULT_INFO flag and the DBX_RESULT_ASSOC flag (which also causes the info property to appear). Or if you did not want associative indices in the data property you could specify both DBX_RESULT_INFO and DBX_RESULT_INDEX but not DBX_RESULT_ASSOC.

Example

```
<? // $dbconn is a dbx connection variable created with dbx_connect
     $dbresult = dbx_query( $dbconn, "SELECT firstName, lastName from People" );
     if ( ! $dbresult ) { echo "There was an error!"; }
     else { ?>
The first first name in the People table is <?= $dbresult->data[0][0] ?><br>
The first last name in the People table is <?= $dbresult->data[0][‘lastName’]
?><br>
This should say ‘firstName’: <?= $dbresult->info[‘name’][0] ?>
This is the SQL type of the lastName field: <?= $dbresult->info[‘type’][1] ?>
There are <?= $dbresult->rows ?> people in the People table.
This should say '2' since we specified two columns: <?= $dbresult->cols ?>
<? // We can use the $dbresult->handle property with any mysql_* function
     //  that takes a result set variable.
    }

// Make a new query but don’t make the info property or the associative indices
because we’re
    expecting a large result set and want to be as effecient as possible:
    $dbresult = dbx_query( $dbconn, “SELECT * from People”, DBX_RESULT_INDEX );
?>
```

dbx_sort

- Re-sorts a result set

$success = dbx_sort($dbresult, $sort_function)

dbx_sort re-sorts the given result set, which was returned from *dbx_query*. The second argument is the name of a function that is used for sorting. The function can be any function, as long as it takes two parameters and returns an integer (-1 if the first parameter is less than the second, 0 if they are equal and 1 if the first is greater than the second).

The function resorts the ‘data’ property of the result set object. In particular, it resorts the order if the rows (the first dimension) of that property. The *dbx_sort* function passes single rows of the ‘data’ property (as an array) as the parameters to the sorting function. Therefore the sorting function should expect arrays as its parameters.

The function returns a true value on success and false in the case of failure.

Example

```
<? // $dbresult is a dbx result set returned by dbx_query

        function sort_lastname( $a, $b ) {
                   if ( $a[‘lastName’] > $b[‘lastName’] ) return 1;
                   else if ($a[‘lastName’] < $b[‘lastName’] ) return -1;
                   else return 0;
        }

         dbx_sort( $dbresult, “sort_lastname” );
          // The $dbresult->data array is now sorted by the last name of the
entries...
?>
```

dbx_compare

- Compare two rows of a dbx result set

$greater_or_less = dbx_compare($row_array_a, $row_array_b, $which_column[, $flags])

dbx_compare compares a single field of two arrays to determine which is greater. This is used with *dbx_sort* to sort a result set. The first two arguments are arrays (usually from the ‘data’ property of a dbx result set). The third argument is the name, or position (first is always 0) of a field in the result set. The function then compares the given field of the two arrays and returns 1 if the first field is greater, -1 if the second field is greater and 0 of the two fields are equal.

Passing in one or more constants as a fourth parameter can modify this default behavior. Multiple constants should be separated by a Boolean-or. There are two different sets of constants; one of each can be specified, if desired:

DBX_CMP_ASC and DBX_CMP_DESC

These flags determine which direction the sorting is performed. Default is DBX_CMP_ASC that performs an ascending sort. The DBX_CMP_DESC flag provides a descending sort.

DBX_CMP_NATIVE, DBX_CMP_TEXT and DBX_CMP_NUMBER

These flags determine how the sorting is performed. The default DBX_CMP_NATIVE performs a straight character-value comparison no matter what the contents of the data are. The DBX_CMP_TEXT flag causes the fields to be compared as alphabetical text using the local character set of the PHP server. The DBX_CMP_NUMBER flag causes the fields to be compared as numbers, with any non-numerical characters ignored.

Example

```
<? //  $dbresult is a dbx result set returned by dbx_query

     // Sort the results by the last name, in descending order, making sure the
values are compared as
     // as text in the local character-set.
     function sort_lastname( $a, $b ) {
                return dbx_compare( $a, $b, "lastName", DBX_DESC|DBX_TEXT );
     }

       dbx_sort( $dbresult, "sort_lastname" );
          // The $dbresult->data array is now sorted by the last name of the
entries...
?>
```