PostgreSQL

Introduction

PostgreSQL database is Open Source product and available without cost. Postgres, developed originally in the UC Berkeley Computer Science Department, pioneered many of the object-relational concepts now becoming available in some commercial databases. It provides SQL92/SQL99 language support, transactions, referential integrity, stored procedures and type extensibility. PostgreSQL is an open source descendant of this original Berkeley code.

Installing/Configuring

Requirements

To use PostgreSQL support, you need PostgreSQL 6.5 or later, PostgreSQL 8.0 or later to enable all PostgreSQL module features. PostgreSQL supports many character encodings including multibyte character encoding. The current version and more information about PostgreSQL is available at » http://www.postgresql.org/ and the » PostgreSQL Documentation.

Installation

In order to enable PostgreSQL support, *--with-pgsql[=DIR]* is required when you compile PHP. DIR is the PostgreSQL base install directory, defaults to */usr/local/pgsql*. If shared object module is available, PostgreSQL module may be loaded using extension directive in *php.ini* or dl() function.

Runtime Configuration

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

PostgreSQL configuration options

Name	Default	Changeable	Changelog
pgsql.allow_persistent	"1"	PHP_INI_SYSTEM	
pgsql.max_persistent	"-1"	PHP_INI_SYSTEM	
pgsql.max_links	"-1"	PHP_INI_SYSTEM	
pgsql.auto_reset_persistent	"0"	PHP_INI_SYSTEM	Available since PHP 4.2.0.
pgsql.ignore_notice	"0"	PHP_INI_ALL	Available since PHP 4.3.0.
pgsql.log_notice	"0"	PHP_INI_ALL	Available since PHP 4.3.0.

For further details and definitions of the PHP_INI_* constants, see the php.ini directives.

Here's a short explanation of the configuration directives.

pgsql.allow_persistent boolean
: Whether to allow persistent Postgres connections.

pgsql.max_persistent integer
: The maximum number of persistent Postgres connections per process.

pgsql.max_links integer
: The maximum number of Postgres connections per process, including persistent connections.

pgsql.auto_reset_persistent integer
: Detect broken persistent links with pg_pconnect(). Needs a little overhead.

pgsql.ignore_notice integer
: Whether or not to ignore PostgreSQL backend notices.

pgsql.log_notice integer
: Whether or not to log PostgreSQL backends notice messages. The PHP directive pgsql.ignore_notice must be off in order to log notice messages.

Resource Types

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

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.

PGSQL_ASSOC (integer)
Passed to pg_fetch_array(). Return an associative array of field names and values.

PGSQL_NUM (integer)
Passed to pg_fetch_array(). Return a numerically indexed array of field numbers and values.

PGSQL_BOTH (integer)
Passed to pg_fetch_array(). Return an array of field values that is both numerically indexed (by field number) and associated (by field name).

PGSQL_CONNECT_FORCE_NEW (integer)
Passed to pg_connect() to force the creation of a new connection, rather then re-using an existing identical connection.

PGSQL_CONNECTION_BAD (integer)
Returned by pg_connection_status() indicating that the database connection is in an invalid state.

PGSQL_CONNECTION_OK (integer)
Returned by pg_connection_status() indicating that the database connection is in a valid state.

PGSQL_SEEK_SET (integer)
Passed to pg_lo_seek(). Seek operation is to begin from the start of the object.

PGSQL_SEEK_CUR (integer)
Passed to pg_lo_seek(). Seek operation is to begin from the current position.

PGSQL_SEEK_END (integer)
Passed to pg_lo_seek(). Seek operation is to begin from the end of the object.

PGSQL_EMPTY_QUERY (integer)
Returned by pg_result_status(). The string sent to the server was empty.

PGSQL_COMMAND_OK (integer)
Returned by pg_result_status(). Successful completion of a command returning no data.

PGSQL_TUPLES_OK (integer)
Returned by pg_result_status(). Successful completion of a command returning data (such as a *SELECT* or *SHOW*).

PGSQL_COPY_OUT (integer)
Returned by pg_result_status(). Copy Out (from server) data transfer started.

PGSQL_COPY_IN (integer)
Returned by pg_result_status(). Copy In (to server) data transfer started.

PGSQL_BAD_RESPONSE (integer)
Returned by pg_result_status(). The server's response was not understood.

PGSQL_NONFATAL_ERROR (integer)
Returned by pg_result_status(). A nonfatal error (a notice or warning) occurred.

PGSQL_FATAL_ERROR (integer)
Returned by pg_result_status(). A fatal error occurred.

PGSQL_TRANSACTION_IDLE (integer)
Returned by pg_transaction_status(). Connection is currently idle, not in a transaction.

PGSQL_TRANSACTION_ACTIVE (integer)
Returned by pg_transaction_status(). A command is in progress on the connection. A query has been sent via the connection and not yet completed.

PGSQL_TRANSACTION_INTRANS (integer)
Returned by pg_transaction_status(). The connection is idle, in a transaction block.

PGSQL_TRANSACTION_INERROR (integer)
Returned by pg_transaction_status(). The connection is idle, in a failed transaction block.

PGSQL_TRANSACTION_UNKNOWN (integer)
Returned by pg_transaction_status(). The connection is bad.

PGSQL_DIAG_SEVERITY (integer)
Passed to pg_result_error_field(). The severity; the field contents are *ERROR*, *FATAL*, or *PANIC* (in an error message), or *WARNING*, *NOTICE*, *DEBUG*, *INFO*, or *LOG* (in a notice message), or a localized translation of one of these. Always present.

PGSQL_DIAG_SQLSTATE (integer)
Passed to pg_result_error_field(). The SQLSTATE code for the error. The SQLSTATE code identifies the type of error that has occurred; it can be used by front-end applications to perform specific operations (such as error handling) in response to a particular database error. This field is not localizable, and is always present.

PGSQL_DIAG_MESSAGE_PRIMARY (integer)
Passed to pg_result_error_field(). The primary human-readable error message (typically one line). Always present.

PGSQL_DIAG_MESSAGE_DETAIL (integer)
Passed to pg_result_error_field(). Detail: an optional secondary error message carrying more detail about the problem. May run to multiple lines.

PGSQL_DIAG_MESSAGE_HINT (integer)
Passed to pg_result_error_field(). Hint: an optional suggestion what to do about the problem. This is intended to differ from detail in that it offers advice (potentially inappropriate) rather than hard facts. May run to multiple lines.

PGSQL_DIAG_STATEMENT_POSITION (integer)
Passed to pg_result_error_field(). A string containing a decimal integer indicating an error cursor position as an index into the original statement string. The first character has index 1, and positions are measured in characters not bytes.

PGSQL_DIAG_INTERNAL_POSITION (integer)
Passed to pg_result_error_field(). This is defined the same as the **PG_DIAG_STATEMENT_POSITION** field, but it is used when the cursor position refers to an internally generated command rather than the one submitted by the client. The **PG_DIAG_INTERNAL_QUERY** field will always appear when this field appears.

PGSQL_DIAG_INTERNAL_QUERY (integer)
Passed to pg_result_error_field(). The text of a failed internally-generated command. This could be, for example, a SQL query issued by a PL/pgSQL function.

PGSQL_DIAG_CONTEXT (integer)
Passed to pg_result_error_field(). An indication of the context in which the error occurred. Presently this includes a call stack traceback of active procedural language functions and internally-generated queries. The trace is one entry per line, most recent first.

PGSQL_DIAG_SOURCE_FILE (integer)
Passed to pg_result_error_field(). The file name of the PostgreSQL source-code location where the error was reported.

PGSQL_DIAG_SOURCE_LINE (integer)
Passed to pg_result_error_field(). The line number of the PostgreSQL source-code location where the error was reported.

PGSQL_DIAG_SOURCE_FUNCTION (integer)
Passed to pg_result_error_field(). The name of the PostgreSQL source-code function reporting the error.

PGSQL_ERRORS_TERSE (integer)
Passed to pg_set_error_verbosity(). Specified that returned messages include severity, primary text, and position only; this will normally fit on a single line.

PGSQL_ERRORS_DEFAULT (integer)
Passed to pg_set_error_verbosity(). The default mode produces messages that include the above plus any detail, hint, or context fields (these may span multiple lines).

PGSQL_ERRORS_VERBOSE (integer)
Passed to pg_set_error_verbosity(). The verbose mode includes all available fields.

PGSQL_STATUS_LONG (integer)
Passed to pg_result_status(). Indicates that numerical result code is desired.

PGSQL_STATUS_STRING (integer)
Passed to pg_result_status(). Indicates that textual result command tag is desired.

PGSQL_CONV_IGNORE_DEFAULT (integer)

Passed to pg_convert(). Ignore default values in the table during conversion.

PGSQL_CONV_FORCE_NULL (integer)
Passed to pg_convert(). Use SQL *NULL* in place of an empty string.

PGSQL_CONV_IGNORE_DEFAULT (integer)
Passed to pg_convert(). Ignore conversion of **NULL** into SQL *NOT NULL* columns.

Examples

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

Example #1 - PostgreSQL extension overview example

```
<?php
// Connecting, selecting database
$dbconn = pg_connect("host=localhost dbname=publishing user=www
password=foo")
   or die('Could not connect: ' . pg_last_error());

// Performing SQL query
$query = 'SELECT * FROM authors';
$result = pg_query($query) or die('Query failed: ' . pg_last_error());

// Printing results in HTML
echo "<table>\n";
while ($line = pg_fetch_array($result, null, PGSQL_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
pg_free_result($result);

// Closing connection
pg_close($dbconn);
?>
```

PostgreSQL Functions

Notes

Note
Not all functions are supported by all builds. It depends on your libpq (The PostgreSQL C client library) version and how libpq is compiled. If PHP PostgreSQL extensions are missing, then it is because your libpq version does not support them.

Note
Most PostgreSQL functions accept `connection` as the first optional parameter. If it is not provided, the last opened connection is used. If it doesn't exist, functions return **FALSE**.

Note
PostgreSQL automatically folds all identifiers (e.g. table/column names) to lower-case values at object creation time and at query time. To force the use of mixed or upper case identifiers, you must escape the identifier using double quotes ("").

Note
PostgreSQL does not have special commands for fetching database schema information (eg. all the tables in the current database). Instead, there is a standard schema named *information_schema* in PostgreSQL 7.4 and above containing system views with all the necessary information, in an easily queryable form. See the » PostgreSQL Documentation for full details.

pg_affected_rows

pg_affected_rows -- Returns number of affected records (tuples)

Description

int **pg_affected_rows** (resource `$result`)

pg_affected_rows() returns the number of tuples (instances/records/rows) affected by *INSERT*, *UPDATE*, and *DELETE* queries.

Note
This function used to be called **pg_cmdtuples()**.

Parameters

`result`
PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

Return Values

The number of rows affected by the query. If no tuple is affected, it will return 0.

Examples

Example #2 - pg_affected_rows() example

```
<?php
$result = pg_query($conn, "INSERT INTO authors VALUES ('Orwell', 2002,
'Animal Farm')");

$cmdtuples = pg_affected_rows($result);

echo $cmdtuples . " tuples are affected.\n";
?>
```

The above example will output:

```
1 tuples are affected.
```

See Also

- pg_query()
- pg_query_params()
- pg_execute()
- pg_num_rows()

pg_cancel_query

pg_cancel_query -- Cancel an asynchronous query

Description

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

pg_cancel_query() cancels an asynchronous query sent with pg_send_query(), pg_send_query_params() or pg_send_execute(). You cannot cancel a query executed using pg_query().

Parameters

`connection`
PostgreSQL database connection resource.

Return Values

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

Examples

Example #3 - pg_cancel_query() example

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");

 if (!pg_connection_busy($dbconn)) {
     pg_send_query($dbconn, "select * from authors; select count(*) from
authors;");
 }

 $res1 = pg_get_result($dbconn);
 echo "First call to pg_get_result(): $res1\n";
 $rows1 = pg_num_rows($res1);
 echo "$res1 has $rows1 records\n\n";

 // Cancel the currently running query.  Will be the second query if it is
 // still running.
 pg_cancel_query($dbconn);
?>
```

The above example will output:

```
First call to pg_get_result(): Resource id #3
Resource id #3 has 3 records
```

See Also

- pg_send_query()
- pg_connection_busy()

pg_client_encoding

pg_client_encoding -- Gets the client encoding

Description

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

PostgreSQL supports automatic character set conversion between server and client for certain character sets. pg_client_encoding() returns the client encoding as a string. The returned string will be one of the standard PostgreSQL encoding identifiers.

Note
This function requires PHP 4.0.3 or higher and PostgreSQL 7.0 or higher. If libpq is compiled without multibyte encoding support, pg_client_encoding() always returns *SQL_ASCII*. Supported encoding depends on PostgreSQL version. Refer to the PostgreSQL Documentation supported encodings. The function used to be called **pg_clientencoding()**.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

The client encoding, or **FALSE** on error.

Examples

Example #4 - pg_client_encoding() example

```
<?php
// Assume $conn is a connection to a ISO-8859-1 database
$encoding = pg_client_encoding($conn);

echo "Client encoding is: ", $encoding, "\n";
?>
```

The above example will output:

```
Client encoding is: ISO-8859-1
```

See Also

- pg_set_client_encoding()

pg_close

pg_close -- Closes a PostgreSQL connection

Description

bool **pg_close** ([resource `$connection`])

pg_close() closes the non-persistent connection to a PostgreSQL database associated with the given *`connection`* resource.

Note
Using pg_close() is not usually necessary, as non-persistent open connections are automatically closed at the end of the script.

If there is open large object resource on the connection, do not close the connection before closing all large object resources.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

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

Examples

Example #5 - pg_close() example

```
<?php
$dbconn = pg_connect("host=localhost port=5432 dbname=mary")
  or die("Could not connect");
echo "Connected successfully";
pg_close($dbconn);
?>
```

The above example will output:

```
Connected successfully
```

See Also

- pg_connect()

pg_connect

pg_connect -- Open a PostgreSQL connection

Description

resource **pg_connect** (string `$connection_string` [, int `$connect_type`])

pg_connect() opens a connection to a PostgreSQL database specified by the *connection_string*.

If a second call is made to pg_connect() with the same *connection_string* as an existing connection, the existing connection will be returned unless you pass **PGSQL_CONNECT_FORCE_NEW** as *connect_type*.

The old syntax with multiple parameters *$conn = pg_connect("host", "port", "options", "tty", "dbname")* has been deprecated.

Parameters

connection_string
: The *connection_string* can be empty to use all default parameters, or it can contain one or more parameter settings separated by whitespace. Each parameter setting is in the form *keyword* = *value*. Spaces around the equal sign are optional. To write an empty value or a value containing spaces, surround it with single quotes, e.g., *keyword* = *'a value'*. Single quotes and backslashes within the value must be escaped with a backslash, i.e., \' and \\. The currently recognized parameter keywords are: *host*, *hostaddr*, *port*, *dbname*, *user*, *password*, *connect_timeout*, *options*, *tty* (ignored), *sslmode*, *requiressl* (deprecated in favor of *sslmode*), and *service*. Which of these arguments exist depends on your PostgreSQL version.

connect_type
: If **PGSQL_CONNECT_FORCE_NEW** is passed, then a new connection is created, even if the *connection_string* is identical to an existing connection.

Return Values

PostgreSQL connection resource on success, **FALSE** on failure.

Examples

Example #6 - Using pg_connect()

```
<?php
$dbconn = pg_connect("dbname=mary");
```

```
//connect to a database named "mary"

$dbconn2 = pg_connect("host=localhost port=5432 dbname=mary");
// connect to a database named "mary" on "localhost" at port "5432"

$dbconn3 = pg_connect("host=sheep port=5432 dbname=mary user=lamb
password=foo");
//connect to a database named "mary" on the host "sheep" with a username and
password

$conn_string = "host=sheep port=5432 dbname=test user=lamb password=bar";
$dbconn4 = pg_connect($conn_string);
//connect to a database named "test" on the host "sheep" with a username and
password
?>
```

See Also

- pg_pconnect()
- pg_close()
- pg_host()
- pg_port()
- pg_tty()
- pg_options()
- pg_dbname()

pg_connection_busy

pg_connection_busy -- Get connection is busy or not

Description

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

pg_connection_busy() determines whether or not a connection is busy. If it is busy, a previous query is still executing. If pg_get_result() is used on the connection, it will be blocked.

Parameters

`connection`
PostgreSQL database connection resource.

Return Values

Returns **TRUE** if the connection is busy, **FALSE** otherwise.

Examples

Example #7 - pg_connection_busy() example

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");
 $bs = pg_connection_busy($dbconn);
 if ($bs) {
     echo 'connection is busy';
 } else {
    echo 'connection is not busy';
 }
?>
```

See Also

- pg_connection_status()
- pg_get_result()

pg_connection_reset

pg_connection_reset -- Reset connection (reconnect)

Description

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

pg_connection_reset() resets the connection. It is useful for error recovery.

Parameters

`connection`
PostgreSQL database connection resource.

Return Values

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

Examples

Example #8 - pg_connection_reset() example

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");
 $dbconn2 = pg_connection_reset($dbconn);
 if ($dbconn2) {
     echo "reset successful\n";
 } else {
     echo "reset failed\n";
 }
?>
```

See Also

- pg_connect()
- pg_pconnect()
- pg_connection_status()

pg_connection_status

pg_connection_status -- Get connection status

Description

int **pg_connection_status** (resource `$connection`)

pg_connection_status() returns the status of the specified *`connection`*.

Parameters

`connection`
PostgreSQL database connection resource.

Return Values

PGSQL_CONNECTION_OK or **PGSQL_CONNECTION_BAD**.

Examples

Example #9 - pg_connection_status() example

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");
 $stat = pg_connection_status($dbconn);
 if ($stat === PGSQL_CONNECTION_OK) {
     echo 'Connection status ok';
 } else {
     echo 'Connection status bad';
 }
?>
```

See Also

- pg_connection_busy()

pg_convert

pg_convert -- Convert associative array values into suitable for SQL statement

Description

array **pg_convert** (resource `$connection`, string `$table_name`, array `$assoc_array` [, int `$options`])

pg_convert() checks and converts the values in *`assoc_array`* into suitable values for use in a SQL statement. Precondition for pg_convert() is the existence of a table *`table_name`* which has at least as many columns as *`assoc_array`* has elements. The fieldnames in *`table_name`* must match the indices in *`assoc_array`* and the corresponding datatypes must be compatible. Returns an array with the converted values on success, **FALSE** otherwise.

Note
If there are boolean fields in *`table_name`* don't use the constant **TRUE** in *`assoc_array`*. It will be converted to the string 'TRUE' which is no valid entry for boolean fields in PostgreSQL. Use one of t, true, 1, y, yes instead.

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Parameters

`connection`
PostgreSQL database connection resource.

`table_name`
Name of the table against which to convert types.

`assoc_array`
Data to be converted.

`options`
Any number of **PGSQL_CONV_IGNORE_DEFAULT**, **PGSQL_CONV_FORCE_NULL** or **PGSQL_CONV_IGNORE_NOT_NULL**, combined.

Return Values

An array of converted values, or **FALSE** on error.

Examples

Example #10 - pg_convert() example

```
<?php
 $dbconn = pg_connect('dbname=foo');

 $tmp = array(
     'author' => 'Joe Thackery',
     'year' => 2005,
     'title' => 'My Life, by Joe Thackery'
 );

 $vals = pg_convert($dbconn, 'authors', $tmp);
?>
```

See Also

- pg_meta_data()

pg_copy_from

pg_copy_from -- Insert records into a table from an array

Description

bool **pg_copy_from** (resource `$connection`, string `$table_name`, array `$rows` [, string `$delimiter` [, string `$null_as`]])

pg_copy_from() inserts records into a table from *rows*. It issues a *COPY FROM* SQL command internally to insert records.

Parameters

connection
: PostgreSQL database connection resource.

table_name
: Name of the table into which to copy the *rows*.

rows
: An array of data to be copied into *table_name*. Each value in *rows* becomes a row in *table_name*. Each value in *rows* should be a delimited string of the values to insert into each field. Values should be linefeed terminated.

delimiter
: The token that separates values for each field in each element of *rows*. Default is *TAB*.

null_as
: How SQL *NULL* values are represented in the *rows*. Default is \N ("\\N").

Return Values

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

Examples

Example #11 - pg_copy_from() example

```
<?php
  $db = pg_connect("dbname=publisher") or die("Could not connect");

  $rows = pg_copy_to($db, $table_name);

  pg_query($db, "DELETE FROM $table_name");
```

```
  pg_copy_from($db, $table_name, $rows);
?>
```

See Also

- pg_copy_to()

pg_copy_to

pg_copy_to -- Copy a table to an array

Description

array **pg_copy_to** (resource `$connection`, string `$table_name` [, string `$delimiter` [, string `$null_as`]])

pg_copy_to() copies a table to an array. It issues *COPY TO* SQL command internally to retrieve records.

Parameters

`connection`
: PostgreSQL database connection resource.

`table_name`
: Name of the table from which to copy the data into *`rows`*.

`delimiter`
: The token that separates values for each field in each element of *`rows`*. Default is *TAB*.

`null_as`
: How SQL *NULL* values are represented in the *`rows`*. Default is \N ("\\N").

Return Values

An array with one element for each line of *COPY* data. It returns **FALSE** on failure.

Examples

Example #12 - pg_copy_to() example

```
<?php
  $db = pg_connect("dbname=publisher") or die("Could not connect");

  $rows = pg_copy_to($db, $table_name);

  pg_query($db, "DELETE FROM $table_name");

  pg_copy_from($db, $table_name, $rows);
?>
```

See Also

- pg_copy_from()

pg_dbname

pg_dbname -- Get the database name

Description

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

pg_dbname() returns the name of the database that the given PostgreSQL *`connection`* resource.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

A string containing the name of the database the *`connection`* is to, or **FALSE** on error.

Examples

Example #13 - pg_dbname() example

```
<?php
 error_reporting(E_ALL);

 pg_connect("host=localhost port=5432 dbname=mary");
 echo pg_dbname(); // mary
?>
```

pg_delete

pg_delete -- Deletes records

Description

mixed **pg_delete** (resource `$connection`, string `$table_name`, array `$assoc_array` [, int `$options`])

pg_delete() deletes records from a table specified by the keys and values in *`assoc_array`*. If *`options`* is specified, pg_convert() is applied to *`assoc_array`* with the specified options.

Parameters

`connection`
PostgreSQL database connection resource.

`table_name`
Name of the table from which to delete rows.

`assoc_array`
An array whose keys are field names in the table *`table_name`*, and whose values are the values of those fields that are to be deleted.

`options`
Any number of **PGSQL_CONV_FORCE_NULL**, **PGSQL_DML_NO_CONV**, **PGSQL_DML_EXEC** or **PGSQL_DML_STRING** combined. If **PGSQL_DML_STRING** is part of the *`options`* then query string is returned.

Return Values

Returns **TRUE** on success or **FALSE** on failure. Returns string if **PGSQL_DML_STRING** is passed via *`options`*.

Examples

Example #14 - pg_delete() example

```
<?php
 $db = pg_connect('dbname=foo');
 // This is safe, since $_POST is converted automatically
 $res = pg_delete($db, 'post_log', $_POST);
 if ($res) {
     echo "POST data is deleted: $res\n";
 } else {
     echo "User must have sent wrong inputs\n";
```

```
 }
?>
```

Warning

This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

See Also

- pg_convert()

pg_end_copy

pg_end_copy -- Sync with PostgreSQL backend

Description

bool **pg_end_copy** ([resource `$connection`])

pg_end_copy() syncs the PostgreSQL frontend (usually a web server process) with the PostgreSQL server after doing a copy operation performed by pg_put_line(). pg_end_copy() must be issued, otherwise the PostgreSQL server may get out of sync with the frontend and will report an error.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

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

Examples

Example #15 - pg_end_copy() example

```
<?php
 $conn = pg_pconnect("dbname=foo");
 pg_query($conn, "create table bar (a int4, b char(16), d float8)");
 pg_query($conn, "copy bar from stdin");
 pg_put_line($conn, "3\thello world\t4.5\n");
 pg_put_line($conn, "4\tgoodbye world\t7.11\n");
 pg_put_line($conn, "\\.\n");
 pg_end_copy($conn);
?>
```

See Also

- pg_put_line()

pg_escape_bytea

pg_escape_bytea -- Escape a string for insertion into a bytea field

Description

string **pg_escape_bytea** ([resource `$connection`], string `$data`)

pg_escape_bytea() escapes string for bytea datatype. It returns escaped string.

Note
When you *SELECT* a bytea type, PostgreSQL returns octal byte values prefixed with '\' (e.g. \032). Users are supposed to convert back to binary format manually. This function requires PostgreSQL 7.2 or later. With PostgreSQL 7.2.0 and 7.2.1, bytea values must be cast when you enable multi-byte support. i.e. *INSERT INTO test_table (image) VALUES ('$image_escaped'::bytea);* PostgreSQL 7.2.2 or later does not need a cast. The exception is when the client and backend character encoding does not match, and there may be multi-byte stream error. User must then cast to bytea to avoid this error.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`data`
: A string containing text or binary data to be inserted into a bytea column.

Return Values

A string containing the escaped data.

ChangeLog

Version	Description
5.2.0	*`connection`* added

Examples

Example #16 - pg_escape_bytea() example

```
<?php
 // Connect to the database
 $dbconn = pg_connect('dbname=foo');

 // Read in a binary file
 $data = file_get_contents('image1.jpg');

 // Escape the binary data
 $escaped = pg_escape_bytea($data);

 // Insert it into the database
 pg_query("INSERT INTO gallery (name, data) VALUES ('Pine trees',
'{$escaped}')");
?>
```

See Also

- pg_unescape_bytea()
- pg_escape_string()

pg_escape_string

pg_escape_string -- Escape a string for insertion into a text field

Description

string **pg_escape_string** ([resource `$connection`], string `$data`)

pg_escape_string() escapes a string for insertion into the database. It returns an escaped string in the PostgreSQL format. Use of this function is recommended instead of addslashes(). If the type of the column is bytea, pg_escape_bytea() must be used instead.

Note
This function requires PostgreSQL 7.2 or later.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`data`
: A string containing text to be escaped.

Return Values

A string containing the escaped data.

ChangeLog

Version	Description
5.2.0	*`connection`* added

Examples

Example #17 - pg_escape_string() example

```
<?php
 // Connect to the database
 $dbconn = pg_connect('dbname=foo');

 // Read in a text file (containing apostrophes and backslashes)
 $data = file_get_contents('letter.txt');

 // Escape the text data
 $escaped = pg_escape_string($data);

 // Insert it into the database
 pg_query("INSERT INTO correspondence (name, data) VALUES ('My letter',
'{$escaped}')");
?>
```

See Also

- pg_escape_bytea()

pg_execute

pg_execute -- Sends a request to execute a prepared statement with given parameters, and waits for the result.

Description

resource **pg_execute** (resource `$connection`, string `$stmtname`, array `$params`)

resource **pg_execute** (string `$stmtname`, array `$params`)

Sends a request to execute a prepared statement with given parameters, and waits for the result.

pg_execute() is like pg_query_params(), but the command to be executed is specified by naming a previously-prepared statement, instead of giving a query string. This feature allows commands that will be used repeatedly to be parsed and planned just once, rather than each time they are executed. The statement must have been prepared previously in the current session. pg_execute() is supported only against PostgreSQL 7.4 or higher connections; it will fail when using earlier versions.

The parameters are identical to pg_query_params(), except that the name of a prepared statement is given instead of a query string.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`stmtname`
: The name of the prepared statement to execute. if "" is specified, then the unnamed statement is executed. The name must have been previously prepared using pg_prepare(), pg_send_prepare() or a *PREPARE* SQL command.

`params`
: An array of parameter values to substitute for the $1, $2, etc. placeholders in the original prepared query string. The number of elements in the array must match the number of placeholders.

Warning
Elements are converted to strings by calling this function.

Return Values

A query result resource on success, or **FALSE** on failure.

Examples

Example #18 - Using pg_execute()

```
<?php
// Connect to a database named "mary"
$dbconn = pg_connect("dbname=mary");

// Prepare a query for execution
$result = pg_prepare($dbconn, "my_query", 'SELECT * FROM shops WHERE name =
$1');

// Execute the prepared query.  Note that it is not necessary to escape
// the string "Joe's Widgets" in any way
$result = pg_execute($dbconn, "my_query", array("Joe's Widgets"));

// Execute the same prepared query, this time with a different parameter
$result = pg_execute($dbconn, "my_query", array("Clothes Clothes Clothes"));

?>
```

See Also

- pg_prepare()
- pg_send_prepare()
- pg_query_params()

pg_fetch_all_columns

pg_fetch_all_columns -- Fetches all rows in a particular result column as an array

Description

array **pg_fetch_all_columns** (resource `$result` [, int `$column`])

pg_fetch_all_columns() returns an array that contains all rows (records) in a particular column of the result resource.

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

Parameters

result
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

column
: Column number, zero-based, to be retrieved from the result resource. Defaults to the first column if not specified.

Return Values

An array with all values in the result column.

FALSE is returned if *column* is larger than the number of columns in the result, or on any other error.

Examples

Example #19 - pg_fetch_all_columns() example

```
<?php
$conn = pg_pconnect("dbname=publisher");
if (!$conn) {
 echo "An error occured.\n";
 exit;
}

$result = pg_query($conn, "SELECT title, name, address FROM authors");
```

```
if (!$result) {
 echo "An error occured.\n";
 exit;
}

// Get an array of all author names
$arr = pg_fetch_all_columns($result, 1);

var_dump($arr);

?>
```

See Also

- pg_fetch_all()

pg_fetch_all

pg_fetch_all -- Fetches all rows from a result as an array

Description

array **pg_fetch_all** (resource `$result`)

pg_fetch_all() returns an array that contains all rows (records) in the result resource.

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

Parameters

result
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

Return Values

An array with all rows in the result. Each row is an array of field values indexed by field name.

FALSE is returned if there are no rows in the result, or on any other error.

Examples

Example #20 - PostgreSQL fetch all

```
<?php
$conn = pg_pconnect("dbname=publisher");
if (!$conn) {
 echo "An error occured.\n";
 exit;
}

$result = pg_query($conn, "SELECT * FROM authors");
if (!$result) {
 echo "An error occured.\n";
 exit;
}
```

```
$arr = pg_fetch_all($result);

var_dump($arr);

?>
```

See Also

- pg_fetch_row()
- pg_fetch_array()
- pg_fetch_object()
- pg_fetch_result()

pg_fetch_array

pg_fetch_array -- Fetch a row as an array

Description

array **pg_fetch_array** (resource `$result` [, int `$row` [, int `$result_type`]])

pg_fetch_array() returns an array that corresponds to the fetched row (record).

pg_fetch_array() is an extended version of pg_fetch_row(). In addition to storing the data in the numeric indices (field number) to the result array, it can also store the data using associative indices (field name). It stores both indicies by default.

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

pg_fetch_array() is NOT significantly slower than using pg_fetch_row(), and is significantly easier to use.

Parameters

`result`
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

`row`
: Row number in result to fetch. Rows are numbered from 0 upwards. If omitted, next row is fetched.

`result_type`
: An optional parameter that controls how the returned array is indexed. *`result_type`* is a constant and can take the following values: **PGSQL_ASSOC**, **PGSQL_NUM** and **PGSQL_BOTH**. Using **PGSQL_NUM**, pg_fetch_array() will return an array with numerical indices, using **PGSQL_ASSOC** it will return only associative indices while **PGSQL_BOTH**, the default, will return both numerical and associative indices.

Return Values

An array indexed numerically (beginning with 0) or associatively (indexed by field name), or both. Each value in the array is represented as a string. Database *NULL* values are returned as **NULL**.

FALSE is returned if *`row`* exceeds the number of rows in the set, there are no more rows,

or on any other error.

ChangeLog

Version	Description
4.1.0	The *row* parameter became optional.
4.0.0	The *result_type* parameter was added.

Examples

Example #21 - pg_fetch_array() example

```
<?php

$conn = pg_pconnect("dbname=publisher");
if (!$conn) {
 echo "An error occured.\n";
 exit;
}

$result = pg_query($conn, "SELECT author, email FROM authors");
if (!$result) {
 echo "An error occured.\n";
 exit;
}

$arr = pg_fetch_array($result, 0, PGSQL_NUM);
echo $arr[0] . " <- Row 1 Author\n";
echo $arr[1] . " <- Row 1 E-mail\n";

// As of PHP 4.1.0, the row parameter is optional; NULL can be passed
instead,
// to pass a result_type.  Successive calls to pg_fetch_array will return
the
// next row.
$arr = pg_fetch_array($result, NULL, PGSQL_ASSOC);
echo $arr["author"] . " <- Row 2 Author\n";
echo $arr["email"] . " <- Row 2 E-mail\n";

$arr = pg_fetch_array($result);
echo $arr["author"] . " <- Row 3 Author\n";
echo $arr[1] . " <- Row 3 E-mail\n";

?>
```

See Also

- pg_fetch_row()
- pg_fetch_object()
- pg_fetch_result()

pg_fetch_assoc

pg_fetch_assoc -- Fetch a row as an associative array

Description

array **pg_fetch_assoc** (resource `$result` [, int `$row`])

pg_fetch_assoc() returns an associative array that corresponds to the fetched row (records).

pg_fetch_assoc() is equivalent to calling pg_fetch_array() with **PGSQL_ASSOC** as the optional third parameter. It only returns an associative array. If you need the numeric indices, use pg_fetch_row().

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

pg_fetch_assoc() is NOT significantly slower than using pg_fetch_row(), and is significantly easier to use.

Parameters

result
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

row
: Row number in result to fetch. Rows are numbered from 0 upwards. If omitted, next row is fetched.

Return Values

An array indexed associatively (by field name). Each value in the array is represented as a string. Database *NULL* values are returned as **NULL**.

FALSE is returned if *row* exceeds the number of rows in the set, there are no more rows, or on any other error.

ChangeLog

Version	Description
4.1.0	The parameter *row* became optional.

Examples

Example #22 - pg_fetch_assoc() example

```
<?php
$conn = pg_connect("dbname=publisher");
if (!$conn) {
 echo "An error occured.\n";
 exit;
}

$result = pg_query($conn, "SELECT id, author, email FROM authors");
if (!$result) {
 echo "An error occured.\n";
 exit;
}

while ($row = pg_fetch_assoc($result)) {
 echo $row['id'];
 echo $row['author'];
 echo $row['email'];
}
?>
```

See Also

- pg_fetch_row()
- pg_fetch_array()
- pg_fetch_object()
- pg_fetch_result()

pg_fetch_object

pg_fetch_object -- Fetch a row as an object

Description

object **pg_fetch_object** (resource `$result` [, int `$row` [, int `$result_type`]])

object **pg_fetch_object** (resource `$result` [, int `$row` [, string `$class_name` [, array `$params`]]])

pg_fetch_object() returns an object with properties that correspond to the fetched row's field names. It can optionally instantiate an object of a specific class, and pass parameters to that class's constructor.

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

Speed-wise, the function is identical to pg_fetch_array(), and almost as fast as pg_fetch_row() (the difference is insignificant).

Parameters

result
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

row
: Row number in result to fetch. Rows are numbered from 0 upwards. If omitted, next row is fetched.

result_type
: Ignored and deprecated. Defaults to **PGSQL_ASSOC**.

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

An object with one attribute for each field name in the result. Database *NULL* values are

returned as **NULL**.

FALSE is returned if *row* exceeds the number of rows in the set, there are no more rows, or on any other error.

ChangeLog

Version	Description
5.0.0	*class_name* and *params* were added. The old form with *result_type* still exists for backwards compatibility.
4.3.0	*result_type* default changed from **PGSQL_BOTH** to **PGSQL_ASSOC**, since the numeric index was illegal.
4.1.0	The parameter *row* became optional.

Examples

Example #23 - pg_fetch_object() example

```
<?php

$database = "store";

$db_conn = pg_connect("host=localhost port=5432 dbname=$database");
if (!$db_conn) {
 echo "Failed connecting to postgres database $database\n";
 exit;
}

$qu = pg_query($db_conn, "SELECT * FROM books ORDER BY author");


while ($data = pg_fetch_object($qu)) {
 echo $data->author . " (";
 echo $data->year . "): ";
 echo $data->title . "<br />";
}

pg_free_result($qu);
pg_close($db_conn);

?>
```

See Also

- pg_query()
- pg_fetch_array()
- pg_fetch_assoc()
- pg_fetch_row()
- pg_fetch_result()

pg_fetch_result

pg_fetch_result -- Returns values from a result resource

Description

string **pg_fetch_result** (resource `$result`, int `$row`, mixed `$field`)

string **pg_fetch_result** (resource `$result`, mixed `$field`)

pg_fetch_result() returns the value of a particular row and field (column) in a PostgreSQL result resource.

Note
This function used to be called **pg_result()**.

Parameters

`result`
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

`row`
: Row number in result to fetch. Rows are numbered from 0 upwards. If omitted, next row is fetched.

`field`
: A string representing the name of the field (column) to fetch, otherwise an int representing the field number to fetch. Fields are numbered from 0 upwards.

Return Values

Boolean is returned as "t" or "f". All other types, including arrays are returned as strings formatted in the same default PostgreSQL manner that you would see in the *psql* program. Database *NULL* values are returned as **NULL**.

FALSE is returned if *`row`* exceeds the number of rows in the set, or on any other error.

Examples

Example #24 - pg_fetch_result() example

```
<?php
$db = pg_connect("dbname=users user=me") || die();

$res = pg_query($db, "SELECT 1 UNION ALL SELECT 2");

$val = pg_fetch_result($res, 1, 0);

echo "First field in the second row is: ", $val, "\n";
?>
```

The above example will output:

```
First field in the second row is: 2
```

See Also

- pg_query()
- pg_fetch_array()

pg_fetch_row

pg_fetch_row -- Get a row as an enumerated array

Description

array **pg_fetch_row** (resource `$result` [, int `$row`])

pg_fetch_row() fetches one row of data from the result associated with the specified *result* resource.

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

Parameters

result
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

row
: Row number in result to fetch. Rows are numbered from 0 upwards. If omitted, next row is fetched.

Return Values

An array, indexed from 0 upwards, with each value represented as a string. Database *NULL* values are returned as **NULL**.

FALSE is returned if *row* exceeds the number of rows in the set, there are no more rows, or on any other error.

ChangeLog

Version	Description
4.1.0	The parameter *row* became optional.

Examples

Example #25 - pg_fetch_row() example

```
<?php

$conn = pg_pconnect("dbname=publisher");
if (!$conn) {
 echo "An error occured.\n";
 exit;
}

$result = pg_query($conn, "SELECT author, email FROM authors");
if (!$result) {
 echo "An error occured.\n";
 exit;
}

while ($row = pg_fetch_row($result)) {
 echo "Author: $row[0]  E-mail: $row[1]";
 echo "<br />\n";
}

?>
```

See Also

- pg_query()
- pg_fetch_array()
- pg_fetch_object()
- pg_fetch_result()

pg_field_is_null

pg_field_is_null -- Test if a field is SQL *NULL*

Description

int **pg_field_is_null** (resource `$result`, int `$row`, mixed `$field`)

int **pg_field_is_null** (resource `$result`, mixed `$field`)

pg_field_is_null() tests if a field in a PostgreSQL result resource is SQL *NULL* or not.

Note
This function used to be called **pg_fieldisnull()**.

Parameters

result
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

row
: Row number in result to fetch. Rows are numbered from 0 upwards. If omitted, current row is fetched.

field
: Field number (starting from 0) as an integer or the field name as a string.

Return Values

Returns *1* if the field in the given row is SQL *NULL*, *0* if not. **FALSE** is returned if the row is out of range, or upon any other error.

Examples

Example #26 - pg_field_is_null() example

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die ("Could not connect");
 $res = pg_query($dbconn, "select * from authors where author = 'Orwell'");
 if ($res) {
     if (pg_field_is_null($res, 0, "year") == 1) {
```

```
          echo "The value of the field year is null.\n";
      }
      if (pg_field_is_null($res, 0, "year") == 0) {
          echo "The value of the field year is not null.\n";
   }
}
?>
```

pg_field_name

pg_field_name -- Returns the name of a field

Description

string **pg_field_name** (resource `$result`, int `$field_number`)

pg_field_name() returns the name of the field occupying the given *`field_number`* in the given PostgreSQL *`result`* resource. Field numbering starts from 0.

Note
This function used to be called **pg_fieldname()**.

Parameters

`result`
PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

`field_number`
Field number, starting from 0.

Return Values

The field name, or **FALSE** on error.

Examples

Example #27 - Getting information about fields

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");

 $res = pg_query($dbconn, "select * from authors where author = 'Orwell'");
 $i = pg_num_fields($res);
 for ($j = 0; $j < $i; $j++) {
     echo "column $j\n";
     $fieldname = pg_field_name($res, $j);
     echo "fieldname: $fieldname\n";
     echo "printed length: " . pg_field_prtlen($res, $fieldname) . "
characters\n";
     echo "storage length: " . pg_field_size($res, $j) . " bytes\n";
```

```
    echo "field type: " . pg_field_type($res, $j) . " \n\n";
 }
?>
```

The above example will output:

```
column 0
fieldname: author
printed length: 6 characters
storage length: -1 bytes
field type: varchar

column 1
fieldname: year
printed length: 4 characters
storage length: 2 bytes
field type: int2

column 2
fieldname: title
printed length: 24 characters
storage length: -1 bytes
field type: varchar
```

See Also

- pg_field_num()

pg_field_num

pg_field_num -- Returns the field number of the named field

Description

int **pg_field_num** (resource `$result`, string `$field_name`)

pg_field_num() will return the number of the field number that corresponds to the *`field_name`* in the given PostgreSQL *`result`* resource.

Note
This function used to be called **pg_fieldnum()**.

Parameters

`result`
PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

`field_name`
The name of the field.

Return Values

The field number (numbered from 0), or -1 on error.

Examples

Example #28 - Getting information about fields

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");

 $res = pg_query($dbconn, "select author, year, title from authors where
author = 'Orwell'");

 echo "Column 'title' is field number: ", pg_field_num($res, 'title');
?>
```

The above example will output:

```
Column 'title' is field number: 2
```

See Also

- pg_field_name()

pg_field_prtlen

pg_field_prtlen -- Returns the printed length

Description

int **pg_field_prtlen** (resource `$result`, int `$row_number`, mixed `$field_name_or_number`)

int **pg_field_prtlen** (resource `$result`, mixed `$field_name_or_number`)

pg_field_prtlen() returns the actual printed length (number of characters) of a specific value in a PostgreSQL *`result`*. Row numbering starts at 0. This function will return -1 on an error.

`field_name_or_number` can be passed either as an integer or as a string. If it is passed as an integer, PHP recognises it as the field number, otherwise as field name.

See the example given at the pg_field_name() page.

Note
This function used to be called **pg_fieldprtlen()**.

Parameters

`result`
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

`row`
: Row number in result. Rows are numbered from 0 upwards. If omitted, current row is fetched.

Return Values

The field printed length, or **FALSE** on error.

Examples

Example #29 - Getting information about fields

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");
```

```
 $res = pg_query($dbconn, "select * from authors where author = 'Orwell'");
 $i = pg_num_fields($res);
 for ($j = 0; $j < $i; $j++) {
     echo "column $j\n";
     $fieldname = pg_field_name($res, $j);
     echo "fieldname: $fieldname\n";
     echo "printed length: " . pg_field_prtlen($res, $fieldname) . "
characters\n";
     echo "storage length: " . pg_field_size($res, $j) . " bytes\n";
     echo "field type: " . pg_field_type($res, $j) . " \n\n";
 }
?>
```

The above example will output:

```
column 0
fieldname: author
printed length: 6 characters
storage length: -1 bytes
field type: varchar

column 1
fieldname: year
printed length: 4 characters
storage length: 2 bytes
field type: int2

column 2
fieldname: title
printed length: 24 characters
storage length: -1 bytes
field type: varchar
```

See Also

- pg_field_size()

pg_field_size

pg_field_size -- Returns the internal storage size of the named field

Description

int **pg_field_size** (resource `$result`, int `$field_number`)

pg_field_size() returns the internal storage size (in bytes) of the field number in the given PostgreSQL *`result`*.

Note
This function used to be called **pg_fieldsize()**.

Parameters

`result`
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

`field_number`
: Field number, starting from 0.

Return Values

The internal field storage size (in bytes). -1 indicates a variable length field. **FALSE** is returned on error.

Examples

Example #30 - Getting information about fields

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");

 $res = pg_query($dbconn, "select * from authors where author = 'Orwell'");
 $i = pg_num_fields($res);
 for ($j = 0; $j < $i; $j++) {
     echo "column $j\n";
     $fieldname = pg_field_name($res, $j);
     echo "fieldname: $fieldname\n";
     echo "printed length: " . pg_field_prtlen($res, $fieldname) . "
characters\n";
```

```
    echo "storage length: " . pg_field_size($res, $j) . " bytes\n";
    echo "field type: " . pg_field_type($res, $j) . " \n\n";
 }
?>
```

The above example will output:

```
column 0
fieldname: author
printed length: 6 characters
storage length: -1 bytes
field type: varchar

column 1
fieldname: year
printed length: 4 characters
storage length: 2 bytes
field type: int2

column 2
fieldname: title
printed length: 24 characters
storage length: -1 bytes
field type: varchar
```

See Also

- pg_field_prtlen()
- pg_field_type()

pg_field_table

pg_field_table -- Returns the name or oid of the tables field

Description

mixed **pg_field_table** (resource `$result`, int `$field_number` [, bool `$oid_only`])

pg_field_table() returns the name of the table that field belongs to, or the table's oid if *`oid_only`* is **TRUE**.

Parameters

`result`
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

`field_number`
: Field number, starting from 0.

`oid_only`
: By default the tables name that field belongs to is returned but if *`oid_only`* is set to **TRUE**, then the oid will instead be returned.

Return Values

On success either the fields table name or oid. Or, **FALSE** on failure.

Examples

Example #31 - Getting table information about a field

```
<?php
$dbconn = pg_connect("dbname=publisher") or die("Could not connect");

$res = pg_query($dbconn, "SELECT bar FROM foo");

echo pg_field_table($res, 0);
echo pg_field_table($res, 0, true);

$res = pg_query($dbconn, "SELECT version()");
var_dump(pg_field_table($res, 0));
?>
```

The above example will output something similar to:

```
foo
14379580

bool(false)
```

Notes

Note
Returning the oid is much faster than returning the table name because fetching the table name requires a query to the database system table.

See Also

- pg_field_name()
- pg_field_type()

pg_field_type_oid

pg_field_type_oid -- Returns the type ID (OID) for the corresponding field number

Description

int **pg_field_type_oid** (resource `$result`, int `$field_number`)

pg_field_type_oid() returns an integer containing the OID of the base type of the given *`field_number`* in the given PostgreSQL *`result`* resource.

You can get more information about the field type by querying PostgreSQL's *pg_type* system table using the OID obtained with this function. The PostgreSQL **format_type()** function will convert a type OID into an SQL standard type name.

Note
If the field uses a PostgreSQL domain (rather than a basic type), it is the OID of the domain's underlying type that is returned, rather than the OID of the domain itself.

Parameters

`result`
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

`field_number`
: Field number, starting from 0.

Return Values

The OID of the field's base type. **FALSE** is returned on error.

Examples

Example #32 - Getting information about fields

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");

 // Assume 'title' is a varchar type
 $res = pg_query($dbconn, "select title from authors where author =
'Orwell'");
```

```
 echo "Title field type OID: ", pg_field_type_oid($res, 0);
?>
```

The above example will output:

```
Title field type OID: 1043
```

See Also

- pg_field_type()
- pg_field_prtlen()
- pg_field_name()

pg_field_type

pg_field_type -- Returns the type name for the corresponding field number

Description

string **pg_field_type** (resource `$result`, int `$field_number`)

pg_field_type() returns a string containing the base type name of the given *`field_number`* in the given PostgreSQL *`result`* resource.

Note
If the field uses a PostgreSQL domain (rather than a basic type), it is the name of the domain's underlying type that is returned, rather than the name of the domain itself.

Note
This function used to be called **pg_fieldtype()**.

Parameters

`result`
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

`field_number`
: Field number, starting from 0.

Return Values

A string containing the base name of the field's type, or **FALSE** on error.

Examples

Example #33 - Getting information about fields

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");

 // Assume 'title' is a varchar type
```

```
 $res = pg_query($dbconn, "select title from authors where author =
'Orwell'");

 echo "Title field type: ", pg_field_type($res, 0);
?>
```

The above example will output:

```
Title field type: varchar
```

See Also

- pg_field_prtlen()
- pg_field_name()
- pg_field_type_oid()

pg_free_result

pg_free_result -- Free result memory

Description

bool **pg_free_result** (resource `$result`)

pg_free_result() frees the memory and data associated with the specified PostgreSQL query result resource.

This function need only be called if memory consumption during script execution is a problem. Otherwise, all result memory will be automatically freed when the script ends.

Note
This function used to be called **pg_freeresult()**.

Parameters

result
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

Return Values

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

Examples

Example #34 - pg_free_result() example

```
<?php
$db = pg_connect("dbname=users user=me") || die();

$res = pg_query($db, "SELECT 1 UNION ALL SELECT 2");

$val = pg_fetch_result($res, 1, 0);

echo "First field in the second row is: ", $val, "\n";

pg_free_result($res);
?>
```

The above example will output:

```
First field in the second row is: 2
```

See Also

- pg_query()
- pg_query_params()
- pg_execute()

pg_get_notify

pg_get_notify -- Gets SQL NOTIFY message

Description

array **pg_get_notify** (resource `$connection` [, int `$result_type`])

pg_get_notify() gets notifications generated by a *NOTIFY* SQL command. To receive notifications, the *LISTEN* SQL command must be issued.

Parameters

`connection`
PostgreSQL database connection resource.

`result_type`
An optional parameter that controls how the returned array is indexed. *`result_type`* is a constant and can take the following values: **PGSQL_ASSOC**, **PGSQL_NUM** and **PGSQL_BOTH**. Using **PGSQL_NUM**, pg_get_notify() will return an array with numerical indices, using **PGSQL_ASSOC** it will return only associative indices while **PGSQL_BOTH**, the default, will return both numerical and associative indices.

Return Values

An array containing the *NOTIFY* message name and backend PID. Otherwise if no *NOTIFY* is waiting, then **FALSE** is returned.

Examples

Example #35 - PostgreSQL NOTIFY message

```
<?php
$conn = pg_pconnect("dbname=publisher");
if (!$conn) {
 echo "An error occured.\n";
 exit;
}

// Listen 'author_updated' message from other processes
pg_query($conn, 'LISTEN author_updated;');
$notify = pg_get_notify($conn);
if (!$notify) {
 echo "No messages\n";
} else {
 print_r($notify);
}
```

```
?>
```

See Also

- pg_get_pid()

pg_get_pid

pg_get_pid -- Gets the backend's process ID

Description

int **pg_get_pid** (resource `$connection`)

pg_get_pid() gets the backend's (database server process) PID. The PID is useful to determine whether or not a *NOTIFY* message received via pg_get_notify() is sent from another process or not.

Parameters

`connection`
PostgreSQL database connection resource.

Return Values

The backend database process ID.

Examples

Example #36 - PostgreSQL backend PID

```
<?php
$conn = pg_pconnect("dbname=publisher");
if (!$conn) {
 echo "An error occured.\n";
 exit;
}

// Backend process PID. Use PID with pg_get_notify()
$pid = pg_get_pid($conn);
?>
```

See Also

- pg_get_notify()

pg_get_result

pg_get_result -- Get asynchronous query result

Description

resource **pg_get_result** ([resource `$connection`])

pg_get_result() gets the result resource from an asynchronous query executed by pg_send_query(), pg_send_query_params() or pg_send_execute().

pg_send_query() and the other asynchronous query functions can send multiple queries to a PostgreSQL server and pg_get_result() is used to get each query's results, one by one.

Parameters

connection
PostgreSQL database connection resource.

Return Values

The result resource, or **FALSE** if no more results are available.

Examples

Example #37 - pg_get_result() example

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");

 if (!pg_connection_busy($dbconn)) {
     pg_send_query($dbconn, "select * from authors; select count(*) from
authors;");
 }

 $res1 = pg_get_result($dbconn);
 echo "First call to pg_get_result(): $res1\n";
 $rows1 = pg_num_rows($res1);
 echo "$res1 has $rows1 records\n\n";

 $res2 = pg_get_result($dbconn);
 echo "Second call to pg_get_result(): $res2\n";
 $rows2 = pg_num_rows($res2);
 echo "$res2 has $rows2 records\n";
?>
```

The above example will output:

```
First call to pg_get_result(): Resource id #3
Resource id #3 has 3 records

Second call to pg_get_result(): Resource id #4
Resource id #4 has 1 records
```

See Also

- pg_send_query()

pg_host

pg_host -- Returns the host name associated with the connection

Description

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

pg_host() returns the host name of the given PostgreSQL *`connection`* resource is connected to.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

A string containing the name of the host the *`connection`* is to, or **FALSE** on error.

Examples

Example #38 - pg_host() example

```
<?php
$pgsql_conn = pg_connect("dbname=mark host=localhost");

if ($pgsql_conn) {
  print "Successfully connected to: " . pg_host($pgsql_conn) . "<br/>\n";
} else {
  print pg_last_error($pgsql_conn);
  exit;
}
?>
```

See Also

- pg_connect()
- pg_pconnect()

pg_insert

pg_insert -- Insert array into table

Description

mixed **pg_insert** (resource `$connection`, string `$table_name`, array `$assoc_array` [, int `$options`])

pg_insert() inserts the values of *`assoc_array`* into the table specified by *`table_name`*. If *`options`* is specified, pg_convert() is applied to *`assoc_array`* with the specified options.

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Parameters

`connection`
: PostgreSQL database connection resource.

`table_name`
: Name of the table into which to insert rows. The table *`table_name`* must at least have as many columns as *`assoc_array`* has elements.

`assoc_array`
: An array whose keys are field names in the table *`table_name`*, and whose values are the values of those fields that are to be inserted.

`options`
: Any number of **PGSQL_CONV_OPTS**, **PGSQL_DML_NO_CONV**, **PGSQL_DML_EXEC**, **PGSQL_DML_ASYNC** or **PGSQL_DML_STRING** combined. If **PGSQL_DML_STRING** is part of the *`options`* then query string is returned.

Return Values

Returns **TRUE** on success or **FALSE** on failure. Returns string if **PGSQL_DML_STRING** is passed via *`options`*.

Examples

Example #39 - pg_insert() example

```
<?php
 $dbconn = pg_connect('dbname=foo');
 // This is safe, since $_POST is converted automatically
 $res = pg_insert($dbconn, 'post_log', $_POST);
 if ($res) {
     echo "POST data is successfully logged\n";
 } else {
     echo "User must have sent wrong inputs\n";
 }
?>
```

See Also

- pg_convert()

pg_last_error

pg_last_error -- Get the last error message string of a connection

Description

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

pg_last_error() returns the last error message for a given *`connection`*.

Error messages may be overwritten by internal PostgreSQL (libpq) function calls. It may not return an appropriate error message if multiple errors occur inside a PostgreSQL module function.

Use pg_result_error(), pg_result_error_field(), pg_result_status() and pg_connection_status() for better error handling.

Note
This function used to be called **pg_errormessage()**.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

A string containing the last error message on the given *`connection`*, or **FALSE** on error.

Examples

Example #40 - pg_last_error() example

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");

 // Query that fails
 $res = pg_query($dbconn, "select * from doesnotexist");

 echo pg_last_error($dbconn);
?>
```

See Also

- pg_result_error()
- pg_result_error_field()

pg_last_notice

pg_last_notice -- Returns the last notice message from PostgreSQL server

Description

string **pg_last_notice** (resource `$connection`)

pg_last_notice() returns the last notice message from the PostgreSQL server on the specified *`connection`*. The PostgreSQL server sends notice messages in several cases, for instance when creating a *SERIAL* column in a table.

With pg_last_notice(), you can avoid issuing useless queries by checking whether or not the notice is related to your transaction.

Notice message tracking can be set to optional by setting 1 for *pgsql.ignore_notice* in *php.ini*.

Notice message logging can be set to optional by setting 0 for *pgsql.log_notice* in *php.ini*. Unless *pgsql.ignore_notice* is set to 0, notice message cannot be logged.

Parameters

`connection`
PostgreSQL database connection resource.

Return Values

A string containing the last notice on the given *`connection`*, or **FALSE** on error.

ChangeLog

Version	Description
4.3.0	This function is now fully implemented. Earlier versions ignores database connection parameter.
4.3.0	The *pgsql.ignore_notice* and *pgsql.log_notice php.ini* directives were added.
4.0.6	PHP 4.0.6 has problem with notice message handling. Use of the PostgreSQL module

	with PHP 4.0.6 is not recommended even if you are not using pg_last_notice().

Examples

Example #41 - pg_last_error() example

```
<?php
 $pgsql_conn = pg_connect("dbname=mark host=localhost");

 $res = pg_query("CREATE TABLE test (id SERIAL)");

 $notice = pg_last_notice($pgsql_conn);

 echo $notice;
?>
```

The above example will output:

```
CREATE TABLE will create implicit sequence "test_id_seq" for "serial" column
"test.id"
```

See Also

- pg_query()
- pg_last_error()

pg_last_oid

pg_last_oid -- Returns the last row's OID

Description

string **pg_last_oid** (resource `$result`)

pg_last_oid() is used to retrieve the *OID* assigned to an inserted row.

OID field became an optional field from PostgreSQL 7.2 and will not be present by default in PostgreSQL 8.1. When the OID field is not present in a table, the programmer must use pg_result_status() to check for successful insertion.

To get the value of a *SERIAL* field in an inserted row, it is necessary to use the PostgreSQL *CURRVAL* function, naming the sequence whose last value is required. If the name of the sequence is unknown, the *pg_get_serial_sequence* PostgreSQL 8.0 function is necessary.

PostgreSQL 8.1 has a function *LASTVAL* that returns the value of the most recently used sequence in the session. This avoids the need for naming the sequence, table or column altogether.

Note
This function used to be called **pg_getlastoid()**.

Parameters

`result`
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

Return Values

A string containing the OID assigned to the most recently inserted row in the specified *`connection`*, or **FALSE** on error or no available OID.

Examples

Example #42 - pg_last_oid() example

```
<?php
```

```
  $pgsql_conn = pg_connect("dbname=mark host=localhost");

  $res1 = pg_query("CREATE TABLE test (a INTEGER) WITH OIDS");

  $res2 = pg_query("INSERT INTO test VALUES (1)");

  $oid = pg_last_oid($res2);
?>
```

See Also

- pg_query()
- pg_result_status()

pg_lo_close

pg_lo_close -- Close a large object

Description

bool **pg_lo_close** (resource `$large_object`)

pg_lo_close() closes a large object. *`large_object`* is a resource for the large object from pg_lo_open().

To use the large object interface, it is necessary to enclose it within a transaction block.

Note
This function used to be called **pg_loclose()**.

Parameters

`result`
PostgreSQL large object (LOB) resource, returned by pg_lo_open().

Return Values

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

Examples

Example #43 - pg_lo_close() example

```
<?php
  $database = pg_connect("dbname=jacarta");
  pg_query($database, "begin");
  $oid = pg_lo_create($database);
  echo "$oid\n";
  $handle = pg_lo_open($database, $oid, "w");
  echo "$handle\n";
  pg_lo_write($handle, "large object data");
  pg_lo_close($handle);
  pg_query($database, "commit");
?>
```

See Also

- pg_lo_open()
- pg_lo_create()
- pg_lo_import()

pg_lo_create

pg_lo_create -- Create a large object

Description

int **pg_lo_create** ([resource `$connection`])

pg_lo_create() creates a large object and returns the *OID* of the large object. PostgreSQL access modes **INV_READ**, **INV_WRITE**, and **INV_ARCHIVE** are not supported, the object is created always with both read and write access. **INV_ARCHIVE** has been removed from PostgreSQL itself (version 6.3 and above).

To use the large object interface, it is necessary to enclose it within a transaction block.

Instead of using the large object interface (which has no access controls and is cumbersome to use), try PostgreSQL's *bytea* column type and pg_escape_bytea().

Note
This function used to be called **pg_locreate()**.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

A large object *OID* or **FALSE** on error.

Examples

Example #44 - pg_lo_create() example

```
<?php
  $database = pg_connect("dbname=jacarta");
  pg_query($database, "begin");
  $oid = pg_lo_create($database);
  echo "$oid\n";
  $handle = pg_lo_open($database, $oid, "w");
```

```
  echo "$handle\n";
  pg_lo_write($handle, "large object data");
  pg_lo_close($handle);
  pg_query($database, "commit");
?>
```

pg_lo_export

pg_lo_export -- Export a large object to file

Description

bool **pg_lo_export** (resource `$connection`, int `$oid`, string `$pathname`)

bool **pg_lo_export** (int `$oid`, string `$pathname`)

pg_lo_export() takes a large object in a PostgreSQL database and saves its contents to a file on the local filesystem.

To use the large object interface, it is necessary to enclose it within a transaction block.

Note
This function used to be called **pg_loexport()**.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`oid`
: The *OID* of the large object in the database.

`pathname`
: The full path and file name of the file in which to write the large object on the client filesystem.

Return Values

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

Examples

Example #45 - pg_lo_export() example

```
<?php
  $database = pg_connect("dbname=jacarta");
```

```
  pg_query($database, "begin");
  $oid = pg_lo_create($database);
  $handle = pg_lo_open($database, $oid, "w");
  pg_lo_write($handle, "large object data");
  pg_lo_close($handle);
  pg_lo_export($database, $oid, '/tmp/lob.dat');
  pg_query($database, "commit");
?>
```

See Also

- pg_lo_import()

pg_lo_import

pg_lo_import -- Import a large object from file

Description

int **pg_lo_import** (resource `$connection`, string `$pathname`)

int **pg_lo_import** (string `$pathname`)

pg_lo_import() creates a new large object in the database using a file on the filesystem as its data source.

To use the large object interface, it is necessary to enclose it within a transaction block.

Note
When safe mode is enabled, PHP checks whether the files or directories being operated upon have the same UID (owner) as the script that is being executed.

Note
This function used to be called **pg_loimport()**.

Parameters

`connection`
PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`pathname`
The full path and file name of the file on the client filesystem from which to read the large object data.

Return Values

The *OID* of the newly created large object, or **FALSE** on failure.

ChangeLog

Version	Description
4.2.0	The syntax of this function changed. It used to be: int **pg_lo_import** (string `$pathname` [, resource `$connection`])

Examples

Example #46 - pg_lo_import() example

```
<?php
  $database = pg_connect("dbname=jacarta");
  pg_query($database, "begin");
  $oid = pg_lo_import($database, '/tmp/lob.dat');
  pg_query($database, "commit");
?>
```

See Also

- pg_lo_export()
- pg_lo_open()

pg_lo_open

pg_lo_open -- Open a large object

Description

resource **pg_lo_open** (resource `$connection`, int `$oid`, string `$mode`)

pg_lo_open() opens a large object in the database and returns large object resource so that it can be manipulated.

Warning
Do not close the database connection before closing the large object resource.

To use the large object interface, it is necessary to enclose it within a transaction block.

Note
This function used to be called **pg_loopen()**.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`oid`
: The *OID* of the large object in the database.

`mode`
: Can be either "r" for read-only, "w" for write only or "rw" for read and write.

Return Values

A large object resource or **FALSE** on error.

Examples

Example #47 - pg_lo_open() example

```
<?php
  $database = pg_connect("dbname=jacarta");
  pg_query($database, "begin");
  $oid = pg_lo_create($database);
  echo "$oid\n";
  $handle = pg_lo_open($database, $oid, "w");
  echo "$handle\n";
  pg_lo_write($handle, "large object data");
  pg_lo_close($handle);
  pg_query($database, "commit");
?>
```

See Also

- pg_lo_close()
- pg_lo_create()

pg_lo_read_all

pg_lo_read_all -- Reads an entire large object and send straight to browser

Description

int **pg_lo_read_all** (resource `$large_object`)

pg_lo_read_all() reads a large object and passes it straight through to the browser after sending all pending headers. Mainly intended for sending binary data like images or sound.

To use the large object interface, it is necessary to enclose it within a transaction block.

Note
This function used to be called **pg_loreadall()**.

Parameters

`large_object`
PostgreSQL large object (LOB) resource, returned by pg_lo_open().

Return Values

Number of bytes read or **FALSE** on error.

Examples

Example #48 - pg_lo_read_all() example

```
<?php
  header('Content-type: image/jpeg');
  $image_oid = 189762345;
  $database = pg_connect("dbname=jacarta");
  pg_query($database, "begin");
  $handle = pg_lo_open($database, $image_oid, "r");
  pg_lo_read_all($handle);
  pg_query($database, "commit");
?>
```

See Also

- pg_lo_read()

pg_lo_read

pg_lo_read -- Read a large object

Description

string **pg_lo_read** (resource `$large_object` [, int `$len`])

pg_lo_read() reads at most *`len`* bytes from a large object and returns it as a string.

To use the large object interface, it is necessary to enclose it within a transaction block.

Note
This function used to be called **pg_loread()**.

Parameters

`large_object`
: PostgreSQL large object (LOB) resource, returned by pg_lo_open().

`len`
: An optional maximum number of bytes to return. Defaults to 8192.

Return Values

A string containing *`len`* bytes from the large object, or **FALSE** on error.

Examples

Example #49 - pg_lo_read() example

```
<?php
  $doc_oid = 189762345;
  $database = pg_connect("dbname=jacarta");
  pg_query($database, "begin");
  $handle = pg_lo_open($database, $doc_oid, "r");
  $data = pg_lo_read($handle, 50000);
  pg_query($database, "commit");
  echo $data;
?>
```

See Also

- pg_lo_read_all()

pg_lo_seek

pg_lo_seek -- Seeks position within a large object

Description

bool **pg_lo_seek** (resource `$large_object`, int `$offset` [, int `$whence`])

pg_lo_seek() seeks a position within a large object resource.

To use the large object interface, it is necessary to enclose it within a transaction block.

Parameters

`large_object`
PostgreSQL large object (LOB) resource, returned by pg_lo_open().

`offset`
The number of bytes to seek.

`whence`
One of the constants **PGSQL_SEEK_SET** (seek from object start), **PGSQL_SEEK_CUR** (seek from current position) or **PGSQL_SEEK_END** (seek from object end) .

Return Values

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

Examples

Example #50 - pg_lo_seek() example

```
<?php
  $doc_oid = 189762345;
  $database = pg_connect("dbname=jacarta");
  pg_query($database, "begin");
  $handle = pg_lo_open($database, $doc_oid, "r");
  // Skip first 50000 bytes
  pg_lo_seek($handle, 50000, PGSQL_SEEK_SET);
  // Read the next 10000 bytes
  $data = pg_lo_read($handle, 10000);
  pg_query($database, "commit");
  echo $data;
?>
```

See Also

- pg_lo_tell()

pg_lo_tell

pg_lo_tell -- Returns current seek position a of large object

Description

int **pg_lo_tell** (resource `$large_object`)

pg_lo_tell() returns the current position (offset from the beginning) of a large object.

To use the large object interface, it is necessary to enclose it within a transaction block.

Parameters

`large_object`
PostgreSQL large object (LOB) resource, returned by pg_lo_open().

Return Values

The current seek offset (in number of bytes) from the beginning of the large object. If there is an error, the return value is negative.

Examples

Example #51 - pg_lo_tell() example

```
<?php
  $doc_oid = 189762345;
  $database = pg_connect("dbname=jacarta");
  pg_query($database, "begin");
  $handle = pg_lo_open($database, $doc_oid, "r");
  // Skip first 50000 bytes
  pg_lo_seek($handle, 50000, PGSQL_SEEK_SET);
  // See how far we've skipped
  $offset = pg_lo_tell($handle);
  echo "Seek position is: $offset";
  pg_query($database, "commit");
?>
```

The above example will output:

```
Seek position is: 50000
```

See Also

- pg_lo_seek()

pg_lo_unlink

pg_lo_unlink -- Delete a large object

Description

bool **pg_lo_unlink** (resource `$connection`, int `$oid`)

pg_lo_unlink() deletes a large object with the *`oid`*. Returns **TRUE** on success or **FALSE** on failure.

To use the large object interface, it is necessary to enclose it within a transaction block.

Note
This function used to be called **pg_lounlink()**.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`oid`
: The *OID* of the large object in the database.

Return Values

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

Examples

Example #52 - pg_lo_unlink() example

```
<?php
  // OID of the large object to delete
  $doc_oid = 189762345;
  $database = pg_connect("dbname=jacarta");
  pg_query($database, "begin");
  pg_lo_unlink($database, $doc_oid);
  pg_query($database, "commit");
?>
```

See Also

- pg_lo_create()
- pg_lo_import()

pg_lo_write

pg_lo_write -- Write to a large object

Description

int **pg_lo_write** (resource `$large_object`, string `$data` [, int `$len`])

pg_lo_write() writes data into a large object at the current seek position.

To use the large object interface, it is necessary to enclose it within a transaction block.

Note
This function used to be called **pg_lowrite()**.

Parameters

`large_object`
PostgreSQL large object (LOB) resource, returned by pg_lo_open().

`data`
The data to be written to the large object. If *`len`* is specified and is less than the length of *`data`*, only *`len`* bytes will be written.

`len`
An optional maximum number of bytes to write. Must be greater than zero and no greater than the length of *`data`*. Defaults to the length of *`data`*.

Return Values

The number of bytes written to the large object, or **FALSE** on error.

Examples

Example #53 - pg_lo_write() example

```
<?php
  $doc_oid = 189762345;
  $data = "This will overwrite the start of the large object.";
  $database = pg_connect("dbname=jacarta");
  pg_query($database, "begin");
  $handle = pg_lo_open($database, $doc_oid, "w");
```

```
  $data = pg_lo_write($handle, $data);
  pg_query($database, "commit");
?>
```

See Also

- pg_lo_create()
- pg_lo_open()

pg_meta_data

pg_meta_data -- Get meta data for table

Description

array **pg_meta_data** (resource `$connection`, string `$table_name`)

pg_meta_data() returns table definition for *table_name* as an array.

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Parameters

`connection`
PostgreSQL database connection resource.

`table_name`
The name of the table.

Return Values

An array of the table definition, or **FALSE** on error.

Examples

Example #54 - Getting table metadata

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");

 $meta = pg_meta_data($dbconn, 'authors');
 if (is_array($meta)) {
     echo '<pre>';
     var_dump($meta);
     echo '</pre>';
 }
?>
```

The above example will output:

```
array(3) {
["author"]=>
array(5) {
 ["num"]=>
 int(1)
 ["type"]=>
 string(7) "varchar"
 ["len"]=>
 int(-1)
 ["not null"]=>
 bool(false)
 ["has default"]=>
 bool(false)
}
["year"]=>
array(5) {
 ["num"]=>
 int(2)
 ["type"]=>
 string(4) "int2"
 ["len"]=>
 int(2)
 ["not null"]=>
 bool(false)
 ["has default"]=>
 bool(false)
}
["title"]=>
array(5) {
 ["num"]=>
 int(3)
 ["type"]=>
 string(7) "varchar"
 ["len"]=>
 int(-1)
 ["not null"]=>
 bool(false)
 ["has default"]=>
 bool(false)
}
}
```

See Also

- pg_convert()

pg_num_fields

pg_num_fields -- Returns the number of fields in a result

Description

int **pg_num_fields** (resource `$result`)

pg_num_fields() returns the number of fields (columns) in a PostgreSQL result resource.

Note
This function used to be called **pg_numfields()**.

Parameters

result
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

Return Values

The number of fields (columns) in the result. On error, -1 is returned.

Examples

Example #55 - pg_num_fields() example

```
<?php
$result = pg_query($conn, "SELECT 1, 2");

$num = pg_num_fields($result);

echo $num . " field(s) returned.\n";
?>
```

The above example will output:

```
2 field(s) returned.
```

See Also

- pg_num_rows()
- pg_affected_rows()

pg_num_rows

pg_num_rows -- Returns the number of rows in a result

Description

int **pg_num_rows** (resource `$result`)

pg_num_rows() will return the number of rows in a PostgreSQL result resource.

Note
This function used to be called **pg_numrows()**.

Parameters

result
: PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

Return Values

The number of rows in the result. On error, -1 is returned.

Examples

Example #56 - pg_num_rows() example

```
<?php
$result = pg_query($conn, "SELECT 1");

$rows = pg_num_rows($result);

echo $rows . " row(s) returned.\n";
?>
```

The above example will output:

```
1 row(s) returned.
```

See Also

- pg_num_fields()
- pg_affected_rows()

pg_options

pg_options -- Get the options associated with the connection

Description

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

pg_options() will return a string containing the options specified on the given PostgreSQL *`connection`* resource.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

A string containing the *`connection`* options, or **FALSE** on error.

Examples

Example #57 - pg_options() example

```
<?php
  $pgsql_conn = pg_connect("dbname=mark host=localhost");
  echo pg_options($pgsql_conn);
?>
```

See Also

- pg_connect()

pg_parameter_status

pg_parameter_status -- Looks up a current parameter setting of the server.

Description

string **pg_parameter_status** (resource `$connection`, string `$param_name`)

string **pg_parameter_status** (string `$param_name`)

Looks up a current parameter setting of the server.

Certain parameter values are reported by the server automatically at connection startup or whenever their values change. pg_parameter_status() can be used to interrogate these settings. It returns the current value of a parameter if known, or **FALSE** if the parameter is not known.

Parameters reported as of PostgreSQL 8.0 include *server_version*, *server_encoding*, *client_encoding*, *is_superuser*, *session_authorization*, *DateStyle*, *TimeZone*, and *integer_datetimes*. (*server_encoding*, *TimeZone*, and *integer_datetimes* were not reported by releases before 8.0.) Note that *server_version*, *server_encoding* and *integer_datetimes* cannot change after PostgreSQL startup.

PostgreSQL 7.3 or lower servers do not report parameter settings, pg_parameter_status() includes logic to obtain values for *server_version* and *client_encoding* anyway. Applications are encouraged to use pg_parameter_status() rather than ad hoc code to determine these values.

Caution
On a pre-7.4 PostgreSQL server, changing *client_encoding* via *SET* after connection startup will not be reflected by pg_parameter_status().

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`param_name`
: Possible *`param_name`* values include *server_version*, *server_encoding*, *client_encoding*, *is_superuser*, *session_authorization*, *DateStyle*, *TimeZone*, and *integer_datetimes*.

Return Values

A string containing the value of the parameter, **FALSE** on failure or invalid *param_name*.

Examples

Example #58 - pg_parameter_status() example

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");

 echo "Server encoding: ", pg_parameter_status($dbconn, "server_encoding");
?>
```

The above example will output:

```
Server encoding: SQL_ASCII
```

pg_pconnect

pg_pconnect -- Open a persistent PostgreSQL connection

Description

resource **pg_pconnect** (string `$connection_string` [, int `$connect_type`])

pg_pconnect() opens a connection to a PostgreSQL database. It returns a connection resource that is needed by other PostgreSQL functions.

If a second call is made to pg_pconnect() with the same *`connection_string`* as an existing connection, the existing connection will be returned unless you pass **PGSQL_CONNECT_FORCE_NEW** as *`connect_type`*.

To enable persistent connection, the pgsql.allow_persistent *php.ini* directive must be set to "On" (which is the default). The maximum number of persistent connection can be defined with the pgsql.max_persistent *php.ini* directive (defaults to -1 for no limit). The total number of connections can be set with the pgsql.max_links *php.ini* directive.

pg_close() will not close persistent links generated by pg_pconnect().

Parameters

`connection_string`
: The *`connection_string`* can be empty to use all default parameters, or it can contain one or more parameter settings separated by whitespace. Each parameter setting is in the form *keyword* = *value*. Spaces around the equal sign are optional. To write an empty value or a value containing spaces, surround it with single quotes, e.g., *keyword* = *'a value'*. Single quotes and backslashes within the value must be escaped with a backslash, i.e., \' and \\. The currently recognized parameter keywords are: *`host`*, *`hostaddr`*, *`port`*, *`dbname`*, *`user`*, *`password`*, *`connect_timeout`*, *`options`*, *`tty`* (ignored), *`sslmode`*, *`requiressl`* (deprecated in favor of *`sslmode`*), and *`service`*. Which of these arguments exist depends on your PostgreSQL version.

`connect_type`
: If **PGSQL_CONNECT_FORCE_NEW** is passed, then a new connection is created, even if the *`connection_string`* is identical to an existing connection.

Return Values

PostgreSQL connection resource on success, **FALSE** on failure.

Examples

Example #59 - Using pg_pconnect()

```
<?php
$dbconn = pg_pconnect("dbname=mary");
//connect to a database named "mary"

$dbconn2 = pg_pconnect("host=localhost port=5432 dbname=mary");
// connect to a database named "mary" on "localhost" at port "5432"

$dbconn3 = pg_pconnect("host=sheep port=5432 dbname=mary user=lamb
password=foo");
//connect to a database named "mary" on the host "sheep" with a username and
password

$conn_string = "host=sheep port=5432 dbname=test user=lamb password=bar";
$dbconn4 = pg_pconnect($conn_string);
//connect to a database named "test" on the host "sheep" with a username and
password
?>
```

See Also

- pg_connect()
- Persistent Database Connections

pg_ping

pg_ping -- Ping database connection

Description

bool **pg_ping** ([resource `$connection`])

pg_ping() pings a database connection and tries to reconnect it if it is broken.

Parameters

`connection`
PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

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

Examples

Example #60 - pg_ping() example

```
<?php
$conn = pg_pconnect("dbname=publisher");
if (!$conn) {
 echo "An error occured.\n";
 exit;
}

if (!pg_ping($conn))
 die("Connection is broken\n");
?>
```

See Also

- pg_connection_status()
- pg_connection_reset()

pg_port

pg_port -- Return the port number associated with the connection

Description

int **pg_port** ([resource `$connection`])

pg_port() returns the port number that the given PostgreSQL *`connection`* resource is connected to.

Parameters

`connection`
PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

An int containing the port number of the database server the *`connection`* is to, or **FALSE** on error.

Examples

Example #61 - pg_port() example

```
<?php
$pgsql_conn = pg_connect("dbname=mark host=localhost");

if ($pgsql_conn) {
  print "Successfully connected to port: " . pg_port($pgsql_conn) .
"<br/>\n";
} else {
  print pg_last_error($pgsql_conn);
  exit;
}
?>
```

pg_prepare

pg_prepare -- Submits a request to create a prepared statement with the given parameters, and waits for completion.

Description

resource **pg_prepare** (resource `$connection`, string `$stmtname`, string `$query`)

resource **pg_prepare** (string `$stmtname`, string `$query`)

pg_prepare() creates a prepared statement for later execution with pg_execute() or pg_send_execute(). This feature allows commands that will be used repeatedly to be parsed and planned just once, rather than each time they are executed. pg_prepare() is supported only against PostgreSQL 7.4 or higher connections; it will fail when using earlier versions.

The function creates a prepared statement named *`stmtname`* from the *`query`* string, which must contain a single SQL command. *`stmtname`* may be "" to create an unnamed statement, in which case any pre-existing unnamed statement is automatically replaced; otherwise it is an error if the statement name is already defined in the current session. If any parameters are used, they are referred to in the *`query`* as $1, $2, etc.

Prepared statements for use with pg_prepare() can also be created by executing SQL *PREPARE* statements. (But pg_prepare() is more flexible since it does not require parameter types to be pre-specified.) Also, although there is no PHP function for deleting a prepared statement, the SQL *DEALLOCATE* statement can be used for that purpose.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`stmtname`
: The name to give the prepared statement. Must be unique per-connection. If "" is specified, then an unnamed statement is created, overwriting any previously defined unnamed statement.

`query`
: The parameterised SQL statement. Must contain only a single statement. (multiple statements separated by semi-colons are not allowed.) If any parameters are used, they are referred to as $1, $2, etc.

Return Values

A query result resource on success, or **FALSE** on failure.

Examples

Example #62 - Using pg_prepare()

```
<?php
// Connect to a database named "mary"
$dbconn = pg_connect("dbname=mary");

// Prepare a query for execution
$result = pg_prepare($dbconn, "my_query", 'SELECT * FROM shops WHERE name =
$1');

// Execute the prepared query.  Note that it is not necessary to escape
// the string "Joe's Widgets" in any way
$result = pg_execute($dbconn, "my_query", array("Joe's Widgets"));

// Execute the same prepared query, this time with a different parameter
$result = pg_execute($dbconn, "my_query", array("Clothes Clothes Clothes"));

?>
```

See Also

- pg_execute()
- pg_send_execute()

pg_put_line

pg_put_line -- Send a NULL-terminated string to PostgreSQL backend

Description

bool **pg_put_line** (string `$data`)

bool **pg_put_line** (resource `$connection`, string `$data`)

pg_put_line() sends a NULL-terminated string to the PostgreSQL backend server. This is needed in conjunction with PostgreSQL's *COPY FROM* command.

COPY is a high-speed data loading interface supported by PostgreSQL. Data is passed in without being parsed, and in a single transaction.

An alternative to using raw pg_put_line() commands is to use pg_copy_from(). This is a far simpler interface.

Note
The application must explicitly send the two characters "\." on the last line to indicate to the backend that it has finished sending its data, before issuing pg_end_copy().

Warning
Use of the pg_put_line() causes most large object operations, including pg_lo_read() and pg_lo_tell(), to subsequently fail. You can use pg_copy_from() and pg_copy_to() instead.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`data`
: A line of text to be sent directly to the PostgreSQL backend. A *NULL* terminator is added automatically.

Return Values

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

Examples

Example #63 - pg_put_line() example

```
<?php
 $conn = pg_pconnect("dbname=foo");
 pg_query($conn, "create table bar (a int4, b char(16), d float8)");
 pg_query($conn, "copy bar from stdin");
 pg_put_line($conn, "3\thello world\t4.5\n");
 pg_put_line($conn, "4\tgoodbye world\t7.11\n");
 pg_put_line($conn, "\\.\n");
 pg_end_copy($conn);
?>
```

See Also

- pg_end_copy()

pg_query_params

pg_query_params -- Submits a command to the server and waits for the result, with the ability to pass parameters separately from the SQL command text.

Description

resource **pg_query_params** (resource `$connection`, string `$query`, array `$params`)

resource **pg_query_params** (string `$query`, array `$params`)

Submits a command to the server and waits for the result, with the ability to pass parameters separately from the SQL command text.

pg_query_params() is like pg_query(), but offers additional functionality: parameter values can be specified separately from the command string proper. pg_query_params() is supported only against PostgreSQL 7.4 or higher connections; it will fail when using earlier versions.

If parameters are used, they are referred to in the *`query`* string as $1, $2, etc. *`params`* specifies the actual values of the parameters. A **NULL** value in this array means the corresponding parameter is SQL *NULL*.

The primary advantage of pg_query_params() over pg_query() is that parameter values may be separated from the *`query`* string, thus avoiding the need for tedious and error-prone quoting and escaping. Unlike pg_query(), pg_query_params() allows at most one SQL command in the given string. (There can be semicolons in it, but not more than one nonempty command.)

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`query`
: The parameterised SQL statement. Must contain only a single statement. (multiple statements separated by semi-colons are not allowed.) If any parameters are used, they are referred to as $1, $2, etc.

`params`
: An array of parameter values to substitute for the $1, $2, etc. placeholders in the original prepared query string. The number of elements in the array must match the number of placeholders.

Return Values

A query result resource on success, or **FALSE** on failure.

Examples

Example #64 - Using pg_query_params()

```
<?php
// Connect to a database named "mary"
$dbconn = pg_connect("dbname=mary");

// Find all shops named Joe's Widgets.  Note that it is not necessary to
// escape "Joe's Widgets"
$result = pg_query_params($dbconn, 'SELECT * FROM shops WHERE name = $1',
array("Joe's Widgets"));

// Compare against just using pg_query
$str = pg_escape_string("Joe's Widgets");
$result = pg_query($dbconn, "SELECT * FROM shops WHERE name = '{$str}'");

?>
```

See Also

- pg_query()

pg_query

pg_query -- Execute a query

Description

resource **pg_query** (string `$query`)

resource **pg_query** (resource `$connection`, string `$query`)

pg_query() executes the *`query`* on the specified database *`connection`*.

If an error occurs, and **FALSE** is returned, details of the error can be retrieved using the pg_last_error() function if the connection is valid.

Note
Although *`connection`* can be omitted, it is not recommended, since it can be the cause of hard to find bugs in scripts.

Note
This function used to be called **pg_exec()**. **pg_exec()** is still available for compatibility reasons, but users are encouraged to use the newer name.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`query`
: The SQL statement or statements to be executed. When multiple statements are passed to the function, they are automatically executed as one transaction, unless there are explicit BEGIN/COMMIT commands included in the query string. However, using multiple transactions in one function call is not recommended.

Return Values

A query result resource on success, or **FALSE** on failure.

Examples

Example #65 - pg_query() example

```
<?php

$conn = pg_pconnect("dbname=publisher");
if (!$conn) {
 echo "An error occured.\n";
 exit;
}

$result = pg_query($conn, "SELECT author, email FROM authors");
if (!$result) {
 echo "An error occured.\n";
 exit;
}

while ($row = pg_fetch_row($result)) {
 echo "Author: $row[0]  E-mail: $row[1]";
 echo "<br />\n";
}

?>
```

Example #66 - Using pg_query() with multiple statements

```
<?php

$conn = pg_pconnect("dbname=publisher");

// these statements will be executed as one transaction

$query = "UPDATE authors SET author=UPPER(author) WHERE id=1;";
$query .= "UPDATE authors SET author=LOWER(author) WHERE id=2;";
$query .= "UPDATE authors SET author=NULL WHERE id=3;";

pg_query($conn, $query);

?>
```

See Also

- pg_connect()
- pg_pconnect()
- pg_fetch_array()

- pg_fetch_object()
- pg_num_rows()
- pg_affected_rows()

pg_result_error_field

pg_result_error_field -- Returns an individual field of an error report.

Description

string **pg_result_error_field** (resource `$result`, int `$fieldcode`)

pg_result_error_field() returns one of the detailed error message fields associated with *`result`* resource. It is only available against a PostgreSQL 7.4 or above server. The error field is specified by the *`fieldcode`*.

Because pg_query() and pg_query_params() return **FALSE** if the query fails, you must use pg_send_query() and pg_get_result() to get the result handle.

If you need to get additional error information from failed pg_query() queries, use pg_set_error_verbosity() and pg_last_error() and then parse the result.

Parameters

`result`
A PostgreSQL query result resource from a previously executed statement.

`fieldcode`
Possible *`fieldcode`* values are: **PGSQL_DIAG_SEVERITY**, **PGSQL_DIAG_SQLSTATE**, **PGSQL_DIAG_MESSAGE_PRIMARY**, **PGSQL_DIAG_MESSAGE_DETAIL**, **PGSQL_DIAG_MESSAGE_HINT**, **PGSQL_DIAG_STATEMENT_POSITION**, **PGSQL_DIAG_INTERNAL_POSITION** (PostgreSQL 8.0+ only), **PGSQL_DIAG_INTERNAL_QUERY** (PostgreSQL 8.0+ only), **PGSQL_DIAG_CONTEXT**, **PGSQL_DIAG_SOURCE_FILE**, **PGSQL_DIAG_SOURCE_LINE** or **PGSQL_DIAG_SOURCE_FUNCTION**.

Return Values

A string containing the contents of the error field, **NULL** if the field does not exist or **FALSE** on failure.

Examples

Example #67 - pg_result_error_field() example

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");

 if (!pg_connection_busy($dbconn)) {
```

```
    pg_send_query($dbconn, "select * from doesnotexist;");
}

$res1 = pg_get_result($dbconn);
echo pg_result_error_field($res1, PGSQL_DIAG_SQLSTATE);
?>
```

See Also

- pg_result_error()

pg_result_error

pg_result_error -- Get error message associated with result

Description

string **pg_result_error** (resource `$result`)

pg_result_error() returns any error message associated with the *`result`* resource. Therefore, the user has a better chance of getting the correct error message than with pg_last_error().

The function pg_result_error_field() can give much greater detail on result errors than pg_result_error().

Because pg_query() returns **FALSE** if the query fails, you must use pg_send_query() and pg_get_result() to get the result handle.

Parameters

`result`
PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

Return Values

Returns a string if there is an error associated with the *`result`* parameter, **FALSE** otherwise.

Examples

Example #68 - pg_result_error() example

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");

 if (!pg_connection_busy($dbconn)) {
     pg_send_query($dbconn, "select * from doesnotexist;");
 }

 $res1 = pg_get_result($dbconn);
 echo pg_result_error($res1);
?>
```

See Also

- pg_result_error_field()
- pg_query()
- pg_send_query()
- pg_get_result()
- pg_last_error()
- pg_last_notice()

pg_result_seek

pg_result_seek -- Set internal row offset in result resource

Description

bool **pg_result_seek** (resource `$result`, int `$offset`)

pg_result_seek() sets the internal row offset in a result resource.

Parameters

`result`
PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

`offset`
Row to move the internal offset to in the *`result`* resource. Rows are numbered starting from zero.

Return Values

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

Examples

Example #69 - pg_result_seek() example

```
<?php

// Connect to the database
$conn = pg_pconnect("dbname=publisher");

// Execute a query
$result = pg_query($conn, "SELECT author, email FROM authors");

// Seek to the 3rd row (assuming there are 3 rows)
pg_result_seek($result, 2);

// Fetch the 3rd row
$row = pg_fetch_row($result);

?>
```

See Also

- pg_fetch_row()
- pg_fetch_assoc()
- pg_fetch_array()
- pg_fetch_object()
- pg_fetch_result()

pg_result_status

pg_result_status -- Get status of query result

Description

mixed **pg_result_status** (resource `$result` [, int `$type`])

pg_result_status() returns the status of a result resource, or the PostgreSQL command completion tag associated with the result

Parameters

`result`
PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

`type`
Either **PGSQL_STATUS_LONG** to return the numeric status of the *`result`*, or **PGSQL_STATUS_STRING** to return the command tag of the *`result`*. If not specified, **PGSQL_STATUS_LONG** is the default.

Return Values

Possible return values are **PGSQL_EMPTY_QUERY**, **PGSQL_COMMAND_OK**, **PGSQL_TUPLES_OK**, **PGSQL_COPY_OUT**, **PGSQL_COPY_IN**, **PGSQL_BAD_RESPONSE**, **PGSQL_NONFATAL_ERROR** and **PGSQL_FATAL_ERROR** if **PGSQL_STATUS_LONG** is specified. Otherwise, a string containing the PostgreSQL command tag is returned.

ChangeLog

Version	Description
4.3.0	The *`type`* parameter was added.

Examples

Example #70 - pg_result_status() example

```
<?php

// Connect to the database
$conn = pg_pconnect("dbname=publisher");

// Execute a COPY
$result = pg_query($conn, "COPY authors FROM STDIN;");

// Get the result status
$status = pg_result_status($result);

// Determine status
if ($status == PGSQL_COPY_IN)
  echo "Copy began.";
else
  echo "Copy failed.";

?>
```

The above example will output:

```
Copy began.
```

See Also

- pg_connection_status()

pg_select

pg_select -- Select records

Description

mixed **pg_select** (resource `$connection`, string `$table_name`, array `$assoc_array` [, int `$options`])

pg_select() selects records specified by *assoc_array* which has *field=>value*. For a successful query, it returns an array containing all records and fields that match the condition specified by *assoc_array*.

If *options* is specified, pg_convert() is applied to *assoc_array* with the specified flags.

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Parameters

`connection`
PostgreSQL database connection resource.

`table_name`
Name of the table from which to select rows.

`assoc_array`
An array whose keys are field names in the table *`table_name`*, and whose values are the conditions that a row must meet to be retrieved.

`options`
Any number of **PGSQL_CONV_FORCE_NULL**, **PGSQL_DML_NO_CONV**, **PGSQL_DML_EXEC**, **PGSQL_DML_ASYNC** or **PGSQL_DML_STRING** combined. If **PGSQL_DML_STRING** is part of the *`options`* then query string is returned.

Return Values

Returns **TRUE** on success or **FALSE** on failure. Returns string if **PGSQL_DML_STRING** is passed via *`options`*.

Examples

Example #71 - pg_select() example

```
<?php
 $db = pg_connect('dbname=foo');
 // This is safe, since $_POST is converted automatically
 $rec = pg_select($db, 'post_log', $_POST);
 if ($rec) {
     echo "Records selected\n";
     var_dump($rec);
 } else {
     echo "User must have sent wrong inputs\n";
 }
?>
```

See Also

- pg_convert()

pg_send_execute

pg_send_execute -- Sends a request to execute a prepared statement with given parameters, without waiting for the result(s).

Description

bool **pg_send_execute** (resource `$connection`, string `$stmtname`, array `$params`)

Sends a request to execute a prepared statement with given parameters, without waiting for the result(s).

This is similar to pg_send_query_params(), but the command to be executed is specified by naming a previously-prepared statement, instead of giving a query string. The function's parameters are handled identically to pg_execute(). Like pg_execute(), it will not work on pre-7.4 versions of PostgreSQL.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`stmtname`
: The name of the prepared statement to execute. if "" is specified, then the unnamed statement is executed. The name must have been previously prepared using pg_prepare(), pg_send_prepare() or a *PREPARE* SQL command.

`params`
: An array of parameter values to substitute for the $1, $2, etc. placeholders in the original prepared query string. The number of elements in the array must match the number of placeholders.

Return Values

Returns **TRUE** on success, **FALSE** on failure. Use pg_get_result() to determine the query result.

Examples

Example #72 - Using pg_send_execute()

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");
```

```
 // Prepare a query for execution
 if (!pg_connection_busy($dbconn)) {
   pg_send_prepare($dbconn, "my_query", 'SELECT * FROM shops WHERE name =
$1');
   $res1 = pg_get_result($dbconn);
 }

 // Execute the prepared query.  Note that it is not necessary to escape
 // the string "Joe's Widgets" in any way
 if (!pg_connection_busy($dbconn)) {
   pg_send_execute($dbconn, "my_query", array("Joe's Widgets"));
   $res2 = pg_get_result($dbconn);
 }

 // Execute the same prepared query, this time with a different parameter
 if (!pg_connection_busy($dbconn)) {
   pg_send_execute($dbconn, "my_query", array("Clothes Clothes Clothes"));
   $res3 = pg_get_result($dbconn);
 }

?>
```

See Also

- pg_prepare()
- pg_send_prepare()
- pg_execute()

pg_send_prepare

pg_send_prepare -- Sends a request to create a prepared statement with the given parameters, without waiting for completion.

Description

bool **pg_send_prepare** (resource `$connection`, string `$stmtname`, string `$query`)

Sends a request to create a prepared statement with the given parameters, without waiting for completion.

This is an asynchronous version of pg_prepare(): it returns **TRUE** if it was able to dispatch the request, and **FALSE** if not. After a successful call, call pg_get_result() to determine whether the server successfully created the prepared statement. The function's parameters are handled identically to pg_prepare(). Like pg_prepare(), it will not work on pre-7.4 versions of PostgreSQL.

Parameters

`connection`
PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`stmtname`
The name to give the prepared statement. Must be unique per-connection. If "" is specified, then an unnamed statement is created, overwriting any previously defined unnamed statement.

`query`
The parameterised SQL statement. Must contain only a single statement. (multiple statements separated by semi-colons are not allowed.) If any parameters are used, they are referred to as $1, $2, etc.

Return Values

Returns **TRUE** on success, **FALSE** on failure. Use pg_get_result() to determine the query result.

Examples

Example #73 - Using pg_send_prepare()

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");

 // Prepare a query for execution
 if (!pg_connection_busy($dbconn)) {
   pg_send_prepare($dbconn, "my_query", 'SELECT * FROM shops WHERE name =
$1');
   $res1 = pg_get_result($dbconn);
 }

 // Execute the prepared query.  Note that it is not necessary to escape
 // the string "Joe's Widgets" in any way
 if (!pg_connection_busy($dbconn)) {
   pg_send_execute($dbconn, "my_query", array("Joe's Widgets"));
   $res2 = pg_get_result($dbconn);
 }

 // Execute the same prepared query, this time with a different parameter
 if (!pg_connection_busy($dbconn)) {
   pg_send_execute($dbconn, "my_query", array("Clothes Clothes Clothes"));
   $res3 = pg_get_result($dbconn);
 }

?>
```

See Also

- pg_connect()
- pg_pconnect()
- pg_execute()
- pg_send_execute()
- pg_send_query_params()

pg_send_query_params

pg_send_query_params -- Submits a command and separate parameters to the server without waiting for the result(s).

Description

bool **pg_send_query_params** (resource `$connection`, string `$query`, array `$params`)

Submits a command and separate parameters to the server without waiting for the result(s).

This is equivalent to pg_send_query() except that query parameters can be specified separately from the *`query`* string. The function's parameters are handled identically to pg_query_params(). Like pg_query_params(), it will not work on pre-7.4 PostgreSQL connections, and it allows only one command in the query string.

Parameters

`connection`
PostgreSQL database connection resource.

`query`
The parameterised SQL statement. Must contain only a single statement. (multiple statements separated by semi-colons are not allowed.) If any parameters are used, they are referred to as $1, $2, etc.

`params`
An array of parameter values to substitute for the $1, $2, etc. placeholders in the original prepared query string. The number of elements in the array must match the number of placeholders.

Return Values

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

Use pg_get_result() to determine the query result.

Examples

Example #74 - Using pg_send_query_params()

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");
```

```
 // Using parameters.  Note that it is not necessary to quote or escape
 // the parameter.
 pg_send_query_params($dbconn, 'select count(*) from authors where city =
$1', array('Perth'));

 // Compare against basic pg_send_query usage
 $str = pg_escape_string('Perth');
 pg_send_query($dbconn, "select count(*) from authors where city =
'${str}'");
?>
```

See Also

- pg_send_query()

pg_send_query

pg_send_query -- Sends asynchronous query

Description

bool **pg_send_query** (resource `$connection`, string `$query`)

pg_send_query() sends a query or queries asynchronously to the *`connection`*. Unlike pg_query(), it can send multiple queries at once to PostgreSQL and get the results one by one using pg_get_result().

Script execution is not blocked while the queries are executing. Use pg_connection_busy() to check if the connection is busy (i.e. the query is executing). Queries may be cancelled using pg_cancel_query().

Although the user can send multiple queries at once, multiple queries cannot be sent over a busy connection. If a query is sent while the connection is busy, it waits until the last query is finished and discards all its results.

Parameters

`connection`
PostgreSQL database connection resource.

`query`
The SQL statement or statements to be executed.

Return Values

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

Use pg_get_result() to determine the query result.

Examples

Example #75 - pg_send_query() example

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");

 if (!pg_connection_busy($dbconn)) {
     pg_send_query($dbconn, "select * from authors; select count(*) from
authors;");
 }
```

```
 $res1 = pg_get_result($dbconn);
 echo "First call to pg_get_result(): $res1\n";
 $rows1 = pg_num_rows($res1);
 echo "$res1 has $rows1 records\n\n";

 $res2 = pg_get_result($dbconn);
 echo "Second call to pg_get_result(): $res2\n";
 $rows2 = pg_num_rows($res2);
 echo "$res2 has $rows2 records\n";
?>
```

The above example will output:

```
First call to pg_get_result(): Resource id #3
Resource id #3 has 3 records

Second call to pg_get_result(): Resource id #4
Resource id #4 has 1 records
```

See Also

- pg_query()
- pg_cancel_query()
- pg_get_result()
- pg_connection_busy()

pg_set_client_encoding

pg_set_client_encoding -- Set the client encoding

Description

int **pg_set_client_encoding** (string `$encoding`)

int **pg_set_client_encoding** (resource `$connection`, string `$encoding`)

pg_set_client_encoding() sets the client encoding and returns 0 if success or -1 if error.

PostgreSQL will automatically convert data in the backend database encoding into the frontend encoding.

Note
The function used to be called **pg_setclientencoding()**.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`encoding`
: The required client encoding. One of *SQL_ASCII*, *EUC_JP*, *EUC_CN*, *EUC_KR*, *EUC_TW*, *UNICODE*, *MULE_INTERNAL*, *LATINX* (X=1...9), *KOI8*, *WIN*, *ALT*, *SJIS*, *BIG5* or *WIN1250*. The exact list of available encodings depends on your PostgreSQL version, so check your PostgreSQL manual for a more specific list.

Return Values

Returns 0 on success or -1 on error.

Examples

Example #76 - pg_set_client_encoding() example

```
<?php

$conn = pg_pconnect("dbname=publisher");
```

```
if (!$conn) {
 echo "An error occured.\n";
 exit;
}

// Set the client encoding to UNICODE.  Data will be automatically
// converted from the backend encoding to the frontend.
pg_set_client_encoding($conn, UNICODE);

$result = pg_query($conn, "SELECT author, email FROM authors");
if (!$result) {
 echo "An error occured.\n";
 exit;
}

// Write out UTF-8 data
while ($row = pg_fetch_row($result)) {
 echo "Author: $row[0]  E-mail: $row[1]";
 echo "<br />\n";
}

?>
```

See Also

- pg_client_encoding()

pg_set_error_verbosity

pg_set_error_verbosity -- Determines the verbosity of messages returned by pg_last_error() and pg_result_error().

Description

int **pg_set_error_verbosity** (resource `$connection`, int `$verbosity`)

int **pg_set_error_verbosity** (int `$verbosity`)

Determines the verbosity of messages returned by pg_last_error() and pg_result_error().

pg_set_error_verbosity() sets the verbosity mode, returning the connection's previous setting. In **PGSQL_ERRORS_TERSE** mode, returned messages include severity, primary text, and position only; this will normally fit on a single line. The default mode (**PGSQL_ERRORS_DEFAULT**) produces messages that include the above plus any detail, hint, or context fields (these may span multiple lines). The **PGSQL_ERRORS_VERBOSE** mode includes all available fields. Changing the verbosity does not affect the messages available from already-existing result objects, only subsequently-created ones.

Parameters

`connection`
PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

`verbosity`
The required verbosity: **PGSQL_ERRORS_TERSE**, **PGSQL_ERRORS_DEFAULT** or **PGSQL_ERRORS_VERBOSE**.

Return Values

The previous verbosity level: **PGSQL_ERRORS_TERSE**, **PGSQL_ERRORS_DEFAULT** or **PGSQL_ERRORS_VERBOSE**.

Examples

Example #77 - pg_set_error_verbosity() example

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");
```

```
 if (!pg_connection_busy($dbconn)) {
     pg_send_query($dbconn, "select * from doesnotexist;");
 }

 pg_set_error_verbosity($dbconn, PGSQL_ERRORS_VERBOSE);
 $res1 = pg_get_result($dbconn);
 echo pg_result_error($res1);
?>
```

See Also

- pg_last_error()
- pg_result_error()

pg_trace

pg_trace -- Enable tracing a PostgreSQL connection

Description

bool **pg_trace** (string `$pathname` [, string `$mode` [, resource `$connection`]])

pg_trace() enables tracing of the PostgreSQL frontend/backend communication to a file. To fully understand the results, one needs to be familiar with the internals of PostgreSQL communication protocol.

For those who are not, it can still be useful for tracing errors in queries sent to the server, you could do for example *grep '^To backend' trace.log* and see what queries actually were sent to the PostgreSQL server. For more information, refer to the » PostgreSQL Documentation.

Parameters

`pathname`
: The full path and file name of the file in which to write the trace log. Same as in fopen().

`pathname`
: An optional file access mode, same as for fopen(). Defaults to "w".

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

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

Examples

Example #78 - pg_trace() example

```
<?php
$pgsql_conn = pg_connect("dbname=mark host=localhost");

if ($pgsql_conn) {
  pg_trace('/tmp/trace.log', 'w', $pgsql_conn);
  pg_query("SELECT 1");
```

```
  pg_untrace($pgsql_conn);
  // Now /tmp/trace.log will contain backend communication
} else {
  print pg_last_error($pgsql_conn);
  exit;
}
?>
```

See Also

- fopen()
- pg_untrace()

pg_transaction_status

pg_transaction_status -- Returns the current in-transaction status of the server.

Description

int **pg_transaction_status** (resource `$connection`)

Returns the current in-transaction status of the server.

Caution
pg_transaction_status() will give incorrect results when using a PostgreSQL 7.3 server that has the parameter *autocommit* set to off. The server-side autocommit feature has been deprecated and does not exist in later server versions.

Parameters

`connection`
PostgreSQL database connection resource.

Return Values

The status can be **PGSQL_TRANSACTION_IDLE** (currently idle),
PGSQL_TRANSACTION_ACTIVE (a command is in progress),
PGSQL_TRANSACTION_INTRANS (idle, in a valid transaction block), or
PGSQL_TRANSACTION_INERROR (idle, in a failed transaction block).
PGSQL_TRANSACTION_UNKNOWN is reported if the connection is bad.
PGSQL_TRANSACTION_ACTIVE is reported only when a query has been sent to the server and not yet completed.

Examples

Example #79 - pg_transaction_status() example

```
<?php
 $dbconn = pg_connect("dbname=publisher") or die("Could not connect");
 $stat = pg_transaction_status($dbconn);
 if ($stat === PGSQL_TRANSACTION_UNKNOWN) {
     echo 'Connection is bad';
 } else if ($stat === PGSQL_TRANSACTION_IDLE) {
     echo 'Connection is currently idle';
 } else {
     echo 'Connection is in a transaction state';
```

```
  }
?>
```

pg_tty

pg_tty -- Return the TTY name associated with the connection

Description

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

pg_tty() returns the TTY name that server side debugging output is sent to on the given PostgreSQL *`connection`* resource.

Note
pg_tty() is obsolete, since the server no longer pays attention to the TTY setting, but the function remains for backwards compatibility.

Parameters

`connection`
: PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

A string containing the debug TTY of the *`connection`*, or **FALSE** on error.

Examples

Example #80 - pg_tty() example

```
<?php
$pgsql_conn = pg_connect("dbname=mark host=localhost");

if ($pgsql_conn) {
  print "Server debug TTY is: " . pg_tty($pgsql_conn) . "<br/>\n";
} else {
  print pg_last_error($pgsql_conn);
  exit;
}
?>
```

pg_unescape_bytea

pg_unescape_bytea -- Unescape binary for bytea type

Description

string **pg_unescape_bytea** (string `$data`)

pg_unescape_bytea() unescapes PostgreSQL bytea data values. It returns the unescaped string, possibly containing binary data.

Note

When you *SELECT* a bytea type, PostgreSQL returns octal byte values prefixed with '\' (e.g. \032). Users are supposed to convert back to binary format manually.

This function requires PostgreSQL 7.2 or later. With PostgreSQL 7.2.0 and 7.2.1, bytea values must be cast when you enable multi-byte support. i.e. *INSERT INTO test_table (image) VALUES ('$image_escaped'::bytea);* PostgreSQL 7.2.2 or later does not need a cast. The exception is when the client and backend character encoding does not match, and there may be multi-byte stream error. User must then cast to bytea to avoid this error.

Parameters

`data`
A string containing PostgreSQL bytea data to be converted into a PHP binary string.

Return Values

A string containing the unescaped data.

Examples

Example #81 - pg_unescape_bytea() example

```
<?php
 // Connect to the database
 $dbconn = pg_connect('dbname=foo');

 // Get the bytea data
 $res = pg_query("SELECT data FROM gallery WHERE name='Pine trees'");
 $raw = pg_fetch_result($res, 'data');
```

```
 // Convert to binary and send to the browser
 header('Content-type: image/jpeg');
 echo pg_unescape_bytea($raw);
?>
```

See Also

- pg_escape_bytea()
- pg_escape_string()

pg_untrace

pg_untrace -- Disable tracing of a PostgreSQL connection

Description

bool **pg_untrace** ([resource `$connection`])

Stop tracing started by pg_trace().

Parameters

`connection`
PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

Always returns **TRUE**.

Examples

Example #82 - pg_untrace() example

```
<?php
$pgsql_conn = pg_connect("dbname=mark host=localhost");

if ($pgsql_conn) {
  pg_trace('/tmp/trace.log', 'w', $pgsql_conn);
  pg_query("SELECT 1");
  pg_untrace($pgsql_conn);
  // Now tracing of backend communication is disabled
} else {
  print pg_last_error($pgsql_conn);
  exit;
}
?>
```

See Also

- pg_trace()

pg_update

pg_update -- Update table

Description

mixed **pg_update** (resource `$connection`, string `$table_name`, array `$data`, array `$condition` [, int `$options`])

pg_update() updates records that matches *condition* with *data*. If *options* is specified, pg_convert() is applied to *data* with specified options.

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Parameters

connection
: PostgreSQL database connection resource.

table_name
: Name of the table into which to update rows.

data
: An array whose keys are field names in the table *table_name*, and whose values are what matched rows are to be updated to.

condition
: An array whose keys are field names in the table *table_name*, and whose values are the conditions that a row must meet to be updated.

options
: Any number of **PGSQL_CONV_OPTS**, **PGSQL_DML_NO_CONV**, **PGSQL_DML_EXEC** or **PGSQL_DML_STRING** combined. If **PGSQL_DML_STRING** is part of the *options* then query string is returned.

Return Values

Returns **TRUE** on success or **FALSE** on failure. Returns string if **PGSQL_DML_STRING** is passed via *options*.

Examples

Example #83 - pg_update() example

```
<?php
 $db = pg_connect('dbname=foo');
 $data = array('field1'=>'AA', 'field2'=>'BB');

 // This is safe, since $_POST is converted automatically
 $res = pg_update($db, 'post_log', $_POST, $data);
 if ($res) {
     echo "Data is updated: $res\n";
 } else {
     echo "User must have sent wrong inputs\n";
 }
?>
```

See Also

- pg_convert()

pg_version

pg_version -- Returns an array with client, protocol and server version (when available)

Description

array **pg_version** ([resource `$connection`])

pg_version() returns an array with the client, protocol and server version. Protocol and server versions are only available if PHP was compiled with PostgreSQL 7.4 or later.

For more detailed server information, use pg_parameter_status().

Parameters

`connection`
PostgreSQL database connection resource. When *`connection`* is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

Returns an array with *client*, *protocol* and *server_version* keys and values (if available). Returns **FALSE** on error or invalid connection.

Examples

Example #84 - pg_version() example

```
<?php
 $dbconn = pg_connect("host=localhost port=5432 dbname=mary")
    or die("Could not connect");

 $v = pg_version($dbconn);

 echo $v['client'];
?>
```

The above example will output:

```
7.4
```

See Also

- pg_parameter_status()