# ODBC (Unified)

# Introduction

In addition to normal ODBC support, the Unified ODBC functions in PHP allow you to access several databases that have borrowed the semantics of the ODBC API to implement their own API. Instead of maintaining multiple database drivers that were all nearly identical, these drivers have been unified into a single set of ODBC functions.

The following databases are supported by the Unified ODBC functions: » Adabas D, » IBM DB2, » iODBC, » Solid, and » Sybase SQL Anywhere.

| **Note** |
| --- |
| With the exception of iODBC, there is no ODBC involved when connecting to the above databases. The functions that you use to speak natively to them just happen to share the same names and syntax as the ODBC functions. However, building PHP with iODBC support enables you to use any ODBC-compliant drivers with your PHP applications. More information on iODBC, is available at » www.iodbc.org with the alternative unixODBC available at » www.unixodbc.org. |

# Installing/Configuring

## Requirements

To access any of the supported databases you need to have the required libraries installed.

## Installation

*--with-adabas[=DIR]*
Include Adabas D support. DIR is the Adabas base install directory, defaults to */usr/local*.

*--with-sapdb[=DIR]*
Include SAP DB support. DIR is SAP DB base install directory, defaults to */usr/local*.

*--with-solid[=DIR]*
Include Solid support. DIR is the Solid base install directory, defaults to */usr/local/solid*.

*--with-ibm-db2[=DIR]*
Include IBM DB2 support. DIR is the DB2 base install directory, defaults to */home/db2inst1/sqllib*.

*--with-empress[=DIR]*
Include Empress support. DIR is the Empress base install directory, defaults to $EMPRESSPATH. From PHP 4, this option only supports Empress Version 8.60 and above.

*--with-empress-bcs[=DIR]*
Include Empress Local Access support. DIR is the Empress base install directory, defaults to $EMPRESSPATH. From PHP 4, this option only supports Empress Version 8.60 and above.

*--with-birdstep[=DIR]*
Include Birdstep support. DIR is the Birdstep base install directory, defaults to */usr/local/birdstep*.

*--with-custom-odbc[=DIR]*
Include a user defined ODBC support. The DIR is ODBC install base directory, which defaults to */usr/local*. Make sure to define CUSTOM_ODBC_LIBS and have some *odbc.h* in your include dirs. E.g., you should define following for Sybase SQL Anywhere 5.5.00 on QNX, prior to run configure script:CPPFLAGS="-DODBC_QNX -DSQLANY_BUG"
LDFLAGS=-lunix
CUSTOM_ODBC_LIBS="-ldblib -lodbc".

*--with-iodbc[=DIR]*
Include iODBC support. DIR is the iODBC base install directory, defaults to */usr/local*.

*--with-esoob[=DIR]*
Include Easysoft OOB support. DIR is the OOB base install directory, defaults to */usr/local/easysoft/oob/client*.

*--with-unixODBC[=DIR]*
Include unixODBC support. DIR is the unixODBC base install directory, defaults to */usr/local*.

*--with-openlink[=DIR]*
Include OpenLink ODBC support. DIR is the OpenLink base install directory, defaults to */usr/local*. This is the same as iODBC.

*--with-dbmaker[=DIR]*
Include DBMaker support. DIR is the DBMaker base install directory, defaults to where the latest version of DBMaker is installed (such as */home/dbmaker/3.6* ).

The Windows version of PHP has built-in support for this extension. You do not need to load any additional extensions in order to use these functions.

## Runtime Configuration

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

**Unified ODBC Configuration Options**

| Name | Default | Changeable | Changelog |
|---|---|---|---|
| odbc.default_db * | NULL | PHP_INI_ALL | |
| odbc.default_user * | NULL | PHP_INI_ALL | |
| odbc.default_pw * | NULL | PHP_INI_ALL | |
| odbc.allow_persistent | "1" | PHP_INI_SYSTEM | |
| odbc.check_persistent | "1" | PHP_INI_SYSTEM | |
| odbc.max_persistent | "-1" | PHP_INI_SYSTEM | |
| odbc.max_links | "-1" | PHP_INI_SYSTEM | |
| odbc.defaultlrl | "4096" | PHP_INI_ALL | |
| | | | |

| | | | |
|---|---|---|---|
| odbc.defaultbinmode | "1" | PHP_INI_ALL | |

| **Note** |
|---|
| Entries marked with * are not implemented yet. |

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

Here's a short explanation of the configuration directives.

*odbc.default_db* string
ODBC data source to use if none is specified in odbc_connect() or odbc_pconnect().

*odbc.default_user* string
User name to use if none is specified in odbc_connect() or odbc_pconnect().

*odbc.default_pw* string
Password to use if none is specified in odbc_connect() or odbc_pconnect().

*odbc.allow_persistent* boolean
Whether to allow persistent ODBC connections.

*odbc.check_persistent* boolean
Check that a connection is still valid before reuse.

*odbc.max_persistent* integer
The maximum number of persistent ODBC connections per process.

*odbc.max_links* integer
The maximum number of ODBC connections per process, including persistent connections.

*odbc.defaultlrl* integer
Handling of LONG fields. Specifies the number of bytes returned to variables. When an integer is used, the value is measured in bytes. Shorthand notation, as described in this FAQ, may also be used.

*odbc.defaultbinmode* integer
Handling of binary data.

## Resource Types

This extension defines two resource types: an ODBC connection identifier and an ODBC result identifier.

# 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.

**ODBC_TYPE** ( integer )

**ODBC_BINMODE_PASSTHRU** ( integer )

**ODBC_BINMODE_RETURN** ( integer )

**ODBC_BINMODE_CONVERT** ( integer )

**SQL_ODBC_CURSORS** ( integer )

**SQL_CUR_USE_DRIVER** ( integer )

**SQL_CUR_USE_IF_NEEDED** ( integer )

**SQL_CUR_USE_ODBC** ( integer )

**SQL_CONCURRENCY** ( integer )

**SQL_CONCUR_READ_ONLY** ( integer )

**SQL_CONCUR_LOCK** ( integer )

**SQL_CONCUR_ROWVER** ( integer )

**SQL_CONCUR_VALUES** ( integer )

**SQL_CURSOR_TYPE** ( integer )

**SQL_CURSOR_FORWARD_ONLY** ( integer )

**SQL_CURSOR_KEYSET_DRIVEN** ( integer )

**SQL_CURSOR_DYNAMIC** ( integer )

**SQL_CURSOR_STATIC** ( integer )

**SQL_KEYSET_SIZE** ( integer )

**SQL_CHAR** ( integer )

**SQL_VARCHAR** ( integer )

**SQL_LONGVARCHAR** ( integer )

**SQL_DECIMAL** ( integer )

**SQL_NUMERIC** ( integer )

**SQL_BIT** ( integer )

**SQL_TINYINT** ( integer )

**SQL_SMALLINT** ( integer )

**SQL_INTEGER** ( integer )

**SQL_BIGINT** ( integer )

**SQL_REAL** ( integer )

**SQL_FLOAT** ( integer )

**SQL_DOUBLE** ( integer )

**SQL_BINARY** ( integer )

**SQL_VARBINARY** ( integer )

**SQL_LONGVARBINARY** ( integer )

**SQL_DATE** ( integer )

**SQL_TIME** ( integer )

**SQL_TIMESTAMP** ( integer )

**SQL_TYPE_DATE** ( integer )

**SQL_TYPE_TIME** ( integer )

**SQL_TYPE_TIMESTAMP** ( integer )

**SQL_BEST_ROWID** ( integer )

**SQL_ROWVER** ( integer )

**SQL_SCOPE_CURROW** ( integer )

**SQL_SCOPE_TRANSACTION** ( integer )

**SQL_SCOPE_SESSION** ( integer )

**SQL_NO_NULLS** ( integer )

**SQL_NULLABLE** ( integer )

**SQL_INDEX_UNIQUE** ( integer )

**SQL_INDEX_ALL** ( integer )

**SQL_ENSURE** ( integer )

**SQL_QUICK** ( integer )

# ODBC Functions

# odbc_autocommit

odbc_autocommit -- Toggle autocommit behaviour

## Description

mixed **odbc_autocommit** ( resource `$connection_id` [, bool `$OnOff` ] )

Toggles autocommit behaviour.

By default, auto-commit is on for a connection. Disabling auto-commit is equivalent with starting a transaction.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`OnOff`*
If *`OnOff`* is **TRUE**, auto-commit is enabled, if it is **FALSE** auto-commit is disabled.

## Return Values

Without the *`OnOff`* parameter, this function returns auto-commit status for *`connection_id`*. Non-zero is returned if auto-commit is on, 0 if it is off, or **FALSE** if an error occurs.

If *`OnOff`* is set, this function returns **TRUE** on success and **FALSE** on failure.

## See Also

- odbc_commit()
- odbc_rollback()

# odbc_binmode

odbc_binmode -- Handling of binary column data

## Description

bool **odbc_binmode** ( resource `$result_id`, int `$mode` )

Enables handling of binary column data. ODBC SQL types affected are BINARY, VARBINARY, and LONGVARBINARY.

When binary SQL data is converted to character C data, each byte (8 bits) of source data is represented as two ASCII characters. These characters are the ASCII character representation of the number in its hexadecimal form. For example, a binary 00000001 is converted to *"01"* and a binary 11111111 is converted to *"FF"*.

### LONGVARBINARY handling

| **binmode** | **longreadlen** | **result** |
|---|---|---|
| ODBC_BINMODE_PASSTHRU | 0 | passthru |
| ODBC_BINMODE_RETURN | 0 | passthru |
| ODBC_BINMODE_CONVERT | 0 | passthru |
| ODBC_BINMODE_PASSTHRU | 0 | passthru |
| ODBC_BINMODE_PASSTHRU | >0 | passthru |
| ODBC_BINMODE_RETURN | >0 | return as is |
| ODBC_BINMODE_CONVERT | >0 | return as char |

If odbc_fetch_into() is used, passthru means that an empty string is returned for these columns.

## Parameters

*`result_id`*
The result identifier. If *`result_id`* is *0*, the settings apply as default for new results.

| **Note** |
| --- |
| Default for longreadlen is *4096* and `mode` defaults to *ODBC_BINMODE_RETURN*. Handling of binary long columns is also affected by odbc_longreadlen(). |

`mode`

Possible values for `mode` are:

- **ODBC_BINMODE_PASSTHRU**: Passthru BINARY data
- **ODBC_BINMODE_RETURN**: Return as is
- **ODBC_BINMODE_CONVERT**: Convert to char and return

## Return Values

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

# odbc_close_all

odbc_close_all -- Close all ODBC connections

## Description

void **odbc_close_all** ( void )

odbc_close_all() will close down all connections to database server(s).

## Parameters

This function has no parameters.

## Return Values

No value is returned.

## Notes

| **Note** |
| --- |
| This function will fail if there are open transactions on a connection. This connection will remain open in this case. |

# odbc_close

odbc_close -- Close an ODBC connection

## Description

void **odbc_close** ( resource `$connection_id` )

Closes down the connection to the database server.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

## Return Values

No value is returned.

## Notes

| **Note** |
| --- |
| This function will fail if there are open transactions on this connection. The connection will remain open in this case. |

# odbc_columnprivileges

odbc_columnprivileges -- Lists columns and associated privileges for the given table

## Description

resource **odbc_columnprivileges** ( resource `$connection_id`, string `$qualifier`, string `$owner`, string `$table_name`, string `$column_name` )

Lists columns and associated privileges for the given table.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`qualifier`*
The qualifier.

*`owner`*
The owner.

*`table_name`*
The table name.

*`column_name`*
The *`column_name`* argument accepts search patterns ('%' to match zero or more characters and '_' to match a single character).

The *`owner`*, *`table_name`*, and *`column_name`* accept search patterns ('%' to match zero or more characters and '_' to match a single character).

## Return Values

Returns an ODBC result identifier or **FALSE** on failure. This result identifier can be used to fetch a list of columns and associated privileges.

The result set has the following columns:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- GRANTOR
- GRANTEE

- PRIVILEGE
- IS_GRANTABLE

The result set is ordered by TABLE_QUALIFIER, TABLE_OWNER and TABLE_NAME.

# odbc_columns

odbc_columns -- Lists the column names in specified tables

## Description

resource **odbc_columns** ( resource `$connection_id` [, string `$qualifier` [, string `$schema` [, string `$table_name` [, string `$column_name` ] ] ] ] )

Lists all columns in the requested range.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`qualifier`*
The qualifier.

*`schema`*
The owner.

*`table_name`*
The table name.

*`column_name`*
The column name.

The *`schema`*, *`table_name`*, and *`column_name`* accept search patterns ('%' to match zero or more characters and '_' to match a single character).

## Return Values

Returns an ODBC result identifier or **FALSE** on failure.

The result set has the following columns:

- TABLE_QUALIFIER
- TABLE_SCHEM
- TABLE_NAME
- COLUMN_NAME
- DATA_TYPE
- TYPE_NAME
- PRECISION

- LENGTH
- SCALE
- RADIX
- NULLABLE
- REMARKS

The result set is ordered by TABLE_QUALIFIER, TABLE_SCHEM and TABLE_NAME.

## See Also

- odbc_columnprivileges() to retrieve associated privileges

# odbc_commit

odbc_commit -- Commit an ODBC transaction

## Description

bool **odbc_commit** ( resource `$connection_id` )

Commits all pending transactions on the connection.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

## Return Values

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

# odbc_connect

odbc_connect -- Connect to a datasource

## Description

resource **odbc_connect** ( string `$dsn`, string `$user`, string `$password` [, int `$cursor_type` ] )

The connection id returned by this functions is needed by other ODBC functions. You can have multiple connections open at once as long as they either use different db or different credentials.

With some ODBC drivers, executing a complex stored procedure may fail with an error similar to: "Cannot open a cursor on a stored procedure that has anything other than a single select statement in it". Using SQL_CUR_USE_ODBC may avoid that error. Also, some drivers don't support the optional row_number parameter in odbc_fetch_row(). SQL_CUR_USE_ODBC might help in that case, too.

## Parameters

*`dsn`*
: The database source name for the connection.

*`user`*
: The username.

*`password`*
: The password.

*`cursor_type`*
: This sets the type of cursor to be used for this connection. This parameter is not normally needed, but can be useful for working around problems with some ODBC drivers. The following constants are defined for cursortype:

- SQL_CUR_USE_IF_NEEDED
- SQL_CUR_USE_ODBC
- SQL_CUR_USE_DRIVER
- SQL_CUR_DEFAULT

## Return Values

Returns an ODBC connection id or 0 ( **FALSE** ) on error.

## See Also

- For persistent connections: odbc_pconnect()

# odbc_cursor

odbc_cursor -- Get cursorname

## Description

string **odbc_cursor** ( resource `$result_id` )

Gets the cursorname for the given result_id.

## Parameters

*`result_id`*
The result identifier.

## Return Values

Returns the cursor name, as a string.

# odbc_data_source

odbc_data_source -- Returns information about a current connection

## Description

array **odbc_data_source** ( resource `$connection_id`, int `$fetch_type` )

This function will return the list of available DNS (after calling it several times).

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`fetch_type`*
The *`fetch_type`* can be one of two constant types: **SQL_FETCH_FIRST**, **SQL_FETCH_NEXT**. Use **SQL_FETCH_FIRST** the first time this function is called, thereafter use the **SQL_FETCH_NEXT**.

## Return Values

Returns **FALSE** on error, and an array upon success.

# odbc_do

odbc_do -- Alias of odbc_exec()

## Description

This function is an alias of: odbc_exec().

# odbc_error

odbc_error -- Get the last error code

## Description

string **odbc_error** ( [ resource `$connection_id` ] )

Returns a six-digit ODBC state, or an empty string if there has been no errors.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

## Return Values

If *`connection_id`* is specified, the last state of that connection is returned, else the last state of any connection is returned.

This function returns meaningful value only if last odbc query failed (i.e. odbc_exec() returned **FALSE** ).

## See Also

- odbc_errormsg()
- odbc_exec()

# odbc_errormsg

odbc_errormsg -- Get the last error message

## Description

string **odbc_errormsg** ( [ resource `$connection_id` ] )

Returns a string containing the last ODBC error message, or an empty string if there has been no errors.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

## Return Values

If *`connection_id`* is specified, the last state of that connection is returned, else the last state of any connection is returned.

This function returns meaningful value only if last odbc query failed (i.e. odbc_exec() returned **FALSE** ).

## See Also

- odbc_error()
- odbc_exec()

# odbc_exec

odbc_exec -- Prepare and execute a SQL statement

## Description

resource **odbc_exec** ( resource `$connection_id`, string `$query_string` [, int `$flags` ] )

Sends an SQL statement to the database server.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`query_string`*
The SQL statement.

*`flags`*

## Return Values

Returns an ODBC result identifier if the SQL command was executed successfully, or **FALSE** on error.

## See Also

- odbc_prepare()
- odbc_execute()

# odbc_execute

odbc_execute -- Execute a prepared statement

## Description

bool **odbc_execute** ( resource `$result_id` [, array `$parameters_array` ] )

Executes a statement prepared with odbc_prepare().

## Parameters

*`result_id`*
The result id resource, from odbc_prepare().

*`parameters_array`*
Parameters in *`parameter_array`* will be substituted for placeholders in the prepared statement in order. Elements of this array will be converted to strings by calling this function. Any parameters in *`parameter_array`* which start and end with single quotes will be taken as the name of a file to read and send to the database server as the data for the appropriate placeholder. If you wish to store a string which actually begins and ends with single quotes, you must add a space or other non-single-quote character to the beginning or end of the parameter, which will prevent the parameter from being taken as a file name. If this is not an option, then you must use another mechanism to store the string, such as executing the query directly with odbc_exec() ).

## Return Values

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

## ChangeLog

| Version | Description |
|---|---|
| 4.2.0 | File reading is now subject to safe mode and open-basedir restrictions in *`parameters_array`*. |
| 4.1.1 | Remote files are no longer supported in *`parameters_array`*. |

# odbc_fetch_array

odbc_fetch_array -- Fetch a result row as an associative array

## Description

array **odbc_fetch_array** ( resource `$result` [, int `$rownumber` ] )

Fetch an associative array from an ODBC query. See the changelog below for when this function is available.

## Parameters

*`result`*
The result resource from odbc_exec().

*`rownumber`*
Optionally choose which row number to retrieve.

## Return Values

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

## ChangeLog

| Version | Description |
|---|---|
| 4.3.3 | This function exists when compiled with IBM DB2 or UnixODBC support. |
| 4.3.2 | This function exists when compiled for Windows. |
| 4.0.2 | This function exists when compiled with DBMaker support. |

## See Also

- odbc_fetch_row()
- odbc_fetch_object()

- odbc_num_rows()

# odbc_fetch_into

odbc_fetch_into -- Fetch one result row into array

## Description

int **odbc_fetch_into** ( resource `$result_id`, array `$result_array` [, int `$rownumber` ] )

Fetch one result row into array.

## Parameters

*`result_id`*
The result resource.

*`result_array`*
The result array that can be of any type since it will be converted to type array. The array will contain the column values starting at array index 0.

*`rownumber`*
The row number.

## Return Values

Returns the number of columns in the result; **FALSE** on error.

## ChangeLog

| Version | Description |
|---|---|
| 4.2.0 | The *`result_array`* and *`rownumber`* parameters have been swapped. This allows the rownumber to be a constant again. |
| 4.0.6 | The *`rownumber`* can no longer be passed in as a constant, but rather as a variable. This again changed in 4.2.0. |
| 4.0.5 | The *`result_array`* parameter no longer needs to be passed in by reference. |

## Examples

**Example #1 - odbc_fetch_into() examples**

```
<?php
$rc = odbc_fetch_into($res_id, $my_array);
?>
```

or

```
<?php
$rc = odbc_fetch_into($res_id, $my_array, 2);
?>
```

# odbc_fetch_object

odbc_fetch_object -- Fetch a result row as an object

## Description

object **odbc_fetch_object** ( resource `$result` [, int `$rownumber` ] )

Fetch an object from an ODBC query. See the changelog below for when this function is available.

## Parameters

*`result`*
The result resource from odbc_exec().

*`rownumber`*
Optionally choose which row number to retrieve.

## Return Values

Returns an object that corresponds to the fetched row, or **FALSE** if there are no more rows.

## ChangeLog

| Version | Description |
|---|---|
| 4.3.3 | This function exists when compiled with IBM DB2 or UnixODBC support. |
| 4.3.2 | This function exists when compiled for Windows. |
| 4.0.2 | This function exists when compiled with DBMaker support. |

## See Also

- odbc_fetch_row()

- odbc_fetch_array()
- odbc_num_rows()

# odbc_fetch_row

odbc_fetch_row -- Fetch a row

## Description

bool **odbc_fetch_row** ( resource `$result_id` [, int `$row_number` ] )

Fetches a row of the data that was returned by odbc_do() or odbc_exec(). After odbc_fetch_row() is called, the fields of that row can be accessed with odbc_result().

## Parameters

*`result_id`*
The result identifier.

*`row_number`*
If *`row_number`* is not specified, odbc_fetch_row() will try to fetch the next row in the result set. Calls to odbc_fetch_row() with and without *`row_number`* can be mixed. To step through the result more than once, you can call odbc_fetch_row() with *`row_number`* 1, and then continue doing odbc_fetch_row() without *`row_number`* to review the result. If a driver doesn't support fetching rows by number, the *`row_number`* parameter is ignored.

## Return Values

Returns **TRUE** if there was a row, **FALSE** otherwise.

# odbc_field_len

odbc_field_len -- Get the length (precision) of a field

## Description

int **odbc_field_len** ( resource `$result_id`, int `$field_number` )

Gets the length of the field referenced by number in the given result identifier.

## Parameters

*`result_id`*
The result identifier.

*`field_number`*
The field number. Field numbering starts at 1.

## Return Values

Returns the field name as a string, or **FALSE** on error.

## See Also

- odbc_field_scale() to get the scale of a floating point number

# odbc_field_name

odbc_field_name -- Get the columnname

## Description

string **odbc_field_name** ( resource `$result_id`, int `$field_number` )

Gets the name of the field occupying the given column number in the given result identifier.

## Parameters

*`result_id`*
The result identifier.

*`field_number`*
The field number. Field numbering starts at 1.

## Return Values

Returns the field name as a string, or **FALSE** on error.

# odbc_field_num

odbc_field_num -- Return column number

## Description

int **odbc_field_num** ( resource `$result_id`, string `$field_name` )

Gets the number of the column slot that corresponds to the named field in the given result identifier.

## Parameters

*`result_id`*
The result identifier.

*`field_name`*
The field name.

## Return Values

Returns the field number as a integer, or **FALSE** on error. Field numbering starts at 1.

# odbc_field_precision

odbc_field_precision -- Alias of odbc_field_len()

## Description

This function is an alias of: odbc_field_len().

## See Also

- odbc_field_scale() to get the scale of a floating point number.

# odbc_field_scale

odbc_field_scale -- Get the scale of a field

## Description

int **odbc_field_scale** ( resource `$result_id`, int `$field_number` )

Gets the scale of the field referenced by number in the given result identifier.

## Parameters

*`result_id`*
The result identifier.

*`field_number`*
The field number. Field numbering starts at 1.

## Return Values

Returns the field scale as a integer, or **FALSE** on error.

# odbc_field_type

odbc_field_type -- Datatype of a field

## Description

string **odbc_field_type** ( resource `$result_id`, int `$field_number` )

Gets the SQL type of the field referenced by number in the given result identifier.

## Parameters

*`result_id`*
The result identifier.

*`field_number`*
The field number. Field numbering starts at 1.

## Return Values

Returns the field type as a string, or **FALSE** on error.

# odbc_foreignkeys

odbc_foreignkeys -- Retrieves a list of foreign keys

## Description

resource **odbc_foreignkeys** ( resource `$connection_id`, string `$pk_qualifier`, string `$pk_owner`, string `$pk_table`, string `$fk_qualifier`, string `$fk_owner`, string `$fk_table` )

Retrieves a list of foreign keys in the specified table or a list of foreign keys in other tables that refer to the primary key in the specified table

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`pk_qualifier`*
The primary key qualifier.

*`pk_owner`*
The primary key owner.

*`pk_table`*
The primary key table.

*`fk_qualifier`*
The foreign key qualifier.

*`fk_owner`*
The foreign key owner.

*`fk_table`*
The foreign key table.

## Return Values

Returns an ODBC result identifier or **FALSE** on failure.

The result set has the following columns:

- PKTABLE_QUALIFIER
- PKTABLE_OWNER
- PKTABLE_NAME
- PKCOLUMN_NAME

- FKTABLE_QUALIFIER
- FKTABLE_OWNER
- FKTABLE_NAME
- FKCOLUMN_NAME
- KEY_SEQ
- UPDATE_RULE
- DELETE_RULE
- FK_NAME
- PK_NAME

If *pk_table* contains a table name, odbc_foreignkeys() returns a result set containing the primary key of the specified table and all of the foreign keys that refer to it.

If *fk_table* contains a table name, odbc_foreignkeys() returns a result set containing all of the foreign keys in the specified table and the primary keys (in other tables) to which they refer.

If both *pk_table* and *fk_table* contain table names, odbc_foreignkeys() returns the foreign keys in the table specified in *fk_table* that refer to the primary key of the table specified in *pk_table*. This should be one key at most.

# odbc_free_result

odbc_free_result -- Free resources associated with a result

## Description

bool **odbc_free_result** ( resource `$result_id` )

Free resources associated with a result.

odbc_free_result() only needs to be called if you are worried about using too much memory while your script is running. All result memory will automatically be freed when the script is finished.

## Parameters

*`result_id`*
The result identifier.

## Return Values

Always returns **TRUE**.

## Notes

| **Note** |
| --- |
| If auto-commit is disabled (see odbc_autocommit() ) and you call odbc_free_result() before committing, all pending transactions are rolled back. |

# odbc_gettypeinfo

odbc_gettypeinfo -- Retrieves information about data types supported by the data source

## Description

resource **odbc_gettypeinfo** ( resource `$connection_id` [, int `$data_type` ] )

Retrieves information about data types supported by the data source.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`data_type`*
The data type, which can be used to restrict the information to a single data type.

## Return Values

Returns an ODBC result identifier or **FALSE** on failure.

The result set has the following columns:

- TYPE_NAME
- DATA_TYPE
- PRECISION
- LITERAL_PREFIX
- LITERAL_SUFFIX
- CREATE_PARAMS
- NULLABLE
- CASE_SENSITIVE
- SEARCHABLE
- UNSIGNED_ATTRIBUTE
- MONEY
- AUTO_INCREMENT
- LOCAL_TYPE_NAME
- MINIMUM_SCALE
- MAXIMUM_SCALE

The result set is ordered by DATA_TYPE and TYPE_NAME.

# odbc_longreadlen

odbc_longreadlen -- Handling of LONG columns

## Description

bool **odbc_longreadlen** ( resource `$result_id`, int `$length` )

Enables handling of LONG and LONGVARBINARY columns.

## Parameters

*`result_id`*
The result identifier.

*`length`*
The number of bytes returned to PHP is controlled by the parameter length. If it is set to 0, Long column data is passed through to the client.

## Return Values

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

## Notes

| **Note** |
| --- |
| Handling of LONGVARBINARY columns is also affected by odbc_binmode(). |

# odbc_next_result

odbc_next_result -- Checks if multiple results are available

## Description

bool **odbc_next_result** ( resource `$result_id` )

Checks if there are more result sets available as well as allowing access to the next result set via odbc_fetch_array(), odbc_fetch_row(), odbc_result(), etc.

## Parameters

*`result_id`*
The result identifier.

## Return Values

Returns **TRUE** if there are more result sets, **FALSE** otherwise.

## Examples

**Example #2 -** odbc_next_result()

```
<?php
$r_Connection = odbc_connect($dsn, $username, $password);

$s_SQL = <<<END_SQL
SELECT 'A'
SELECT 'B'
SELECT 'C'
END_SQL;

$r_Results = odbc_exec($r_Connection, $s_SQL);

$a_Row1 = odbc_fetch_array($r_Results);
$a_Row2 = odbc_fetch_array($r_Results);
echo "Dump first result set";
var_dump($a_Row1, $a_Row2);

echo "Get second results set ";
var_dump(odbc_next_result($r_Results));

$a_Row1 = odbc_fetch_array($r_Results);
$a_Row2 = odbc_fetch_array($r_Results);
echo "Dump second result set ";
var_dump($a_Row1, $a_Row2);
```

```
echo "Get third results set ";
var_dump(odbc_next_result($r_Results));

$a_Row1 = odbc_fetch_array($r_Results);
$a_Row2 = odbc_fetch_array($r_Results);
echo "Dump third result set ";
var_dump($a_Row1, $a_Row2);

echo "Try for a fourth result set ";
var_dump(odbc_next_result($r_Results));
?>
```

The above example will output:

```
Dump first result set array(1) {
 ["A"]=>
 string(1) "A"
}
bool(false)
Get second results set bool(true)
Dump second result set array(1) {
 ["B"]=>
 string(1) "B"
}
bool(false)
Get third results set bool(true)
Dump third result set array(1) {
 ["C"]=>
 string(1) "C"
}
bool(false)
Try for a fourth result set bool(false)
```

# odbc_num_fields

odbc_num_fields -- Number of columns in a result

## Description

int **odbc_num_fields** ( resource `$result_id` )

Gets the number of fields (columns) in an ODBC result.

## Parameters

*`result_id`*
The result identifier returned by odbc_exec().

## Return Values

Returns the number of fields, or -1 on error.

# odbc_num_rows

odbc_num_rows -- Number of rows in a result

## Description

int **odbc_num_rows** ( resource `$result_id` )

Gets the number of rows in a result. For INSERT, UPDATE and DELETE statements odbc_num_rows() returns the number of rows affected. For a SELECT clause this *can* be the number of rows available.

## Parameters

*`result_id`*
The result identifier returned by odbc_exec().

## Return Values

Returns the number of rows in an ODBC result. This function will return -1 on error.

## Notes

| **Note** |
| --- |
| Using odbc_num_rows() to determine the number of rows available after a SELECT will return -1 with many drivers. |

# odbc_pconnect

odbc_pconnect -- Open a persistent database connection

## Description

resource **odbc_pconnect** ( string `$dsn`, string `$user`, string `$password` [, int `$cursor_type` ] )

Opens a persistent database connection.

This function is much like odbc_connect(), except that the connection is not really closed when the script has finished. Future requests for a connection with the same *`dsn`*, *`user`*, *`password`* combination (via odbc_connect() and odbc_pconnect() ) can reuse the persistent connection.

## Parameters

See odbc_connect() for details.

## Return Values

Returns an ODBC connection id or 0 ( **FALSE** ) on error.

## Notes

| **Note** |
| --- |
| Persistent connections have no effect if PHP is used as a CGI program. |

## See Also

- odbc_connect()
- Persistent Database Connections

# odbc_prepare

odbc_prepare -- Prepares a statement for execution

## Description

resource **odbc_prepare** ( resource `$connection_id`, string `$query_string` )

Prepares a statement for execution. The result identifier can be used later to execute the statement with odbc_execute().

Some databases (such as IBM DB2, MS SQL Server, and Oracle) support stored procedures that accept parameters of type IN, INOUT, and OUT as defined by the ODBC specification. However, the Unified ODBC driver currently only supports parameters of type IN to stored procedures.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`query_string`*
The query string statement being prepared.

## Return Values

Returns an ODBC result identifier if the SQL command was prepared successfully. Returns **FALSE** on error.

## Examples

**Example #3 - odbc_prepare() example**

In the following code, *$res* will only be valid if all three parameters to myproc are IN parameters:

```
<?php
$a = 1;
$b = 2;
$c = 3;
$stmt = odbc_prepare($conn, 'CALL myproc(?,?,?)');
$res  = odbc_execute($stmt, array($a, $b, $c));
?>
```

If you need to call a stored procedure using INOUT or OUT parameters, the recommended workaround is to use a native extension for your database (for example, mssql for MS SQL Server, or oci8 for Oracle).

# odbc_primarykeys

odbc_primarykeys -- Gets the primary keys for a table

## Description

resource **odbc_primarykeys** ( resource `$connection_id`, string `$qualifier`, string `$owner`, string `$table` )

Returns a result identifier that can be used to fetch the column names that comprise the primary key for a table.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`qualifier`*

*`owner`*

*`table`*

## Return Values

Returns an ODBC result identifier or **FALSE** on failure.

The result set has the following columns:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- COLUMN_NAME
- KEY_SEQ
- PK_NAME

# odbc_procedurecolumns

odbc_procedurecolumns -- Retrieve information about parameters to procedures

## Description

resource **odbc_procedurecolumns** ( resource `$connection_id` )

resource **odbc_procedurecolumns** ( resource `$connection_id`, string `$qualifier`, string `$owner`, string `$proc`, string `$column` )

Retrieve information about parameters to procedures.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`qualifier`*
The qualifier.

*`owner`*
The owner. This parameter accepts the following search patterns: "%" to match zero or more characters, and "_" to match a single character.

*`proc`*
The proc. This parameter accepts the following search patterns: "%" to match zero or more characters, and "_" to match a single character.

*`column`*
The column. This parameter accepts the following search patterns: "%" to match zero or more characters, and "_" to match a single character.

## Return Values

Returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures. Returns an ODBC result identifier or **FALSE** on failure.

The result set has the following columns:

- PROCEDURE_QUALIFIER
- PROCEDURE_OWNER
- PROCEDURE_NAME
- COLUMN_NAME

- COLUMN_TYPE
- DATA_TYPE
- TYPE_NAME
- PRECISION
- LENGTH
- SCALE
- RADIX
- NULLABLE
- REMARKS

The result set is ordered by PROCEDURE_QUALIFIER, PROCEDURE_OWNER, PROCEDURE_NAME and COLUMN_TYPE.

# odbc_procedures

odbc_procedures -- Get the list of procedures stored in a specific data source

## Description

resource **odbc_procedures** ( resource `$connection_id` )

resource **odbc_procedures** ( resource `$connection_id`, string `$qualifier`, string `$owner`, string `$name` )

Lists all procedures in the requested range.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`qualifier`*
The qualifier.

*`owner`*
The owner. This parameter accepts the following search patterns: "%" to match zero or more characters, and "_" to match a single character.

*`name`*
The name. This parameter accepts the following search patterns: "%" to match zero or more characters, and "_" to match a single character.

## Return Values

Returns an ODBC result identifier containing the information or **FALSE** on failure.

The result set has the following columns:

- PROCEDURE_QUALIFIER
- PROCEDURE_OWNER
- PROCEDURE_NAME
- NUM_INPUT_PARAMS
- NUM_OUTPUT_PARAMS
- NUM_RESULT_SETS
- REMARKS
- PROCEDURE_TYPE

# odbc_result_all

odbc_result_all -- Print result as HTML table

## Description

int **odbc_result_all** ( resource `$result_id` [, string `$format` ] )

Prints all rows from a result identifier produced by odbc_exec(). The result is printed in HTML table format.

## Parameters

*`result_id`*
The result identifier.

*`format`*
Additional overall table formatting.

## Return Values

Returns the number of rows in the result or **FALSE** on error.

# odbc_result

odbc_result -- Get result data

## Description

mixed **odbc_result** ( resource `$result_id`, mixed `$field` )

Get result data

## Parameters

*`result_id`*
The ODBC resource.

*`field`*
The field name being retrieved. It can either be an integer containing the column number of the field you want; or it can be a string containing the name of the field.

## Return Values

Returns the string contents of the field, **FALSE** on error, **NULL** for NULL data, or **TRUE** for binary data.

## Examples

The first call to odbc_result() returns the value of the third field in the current record of the query result. The second function call to odbc_result() returns the value of the field whose field name is "val" in the current record of the query result. An error occurs if a column number parameter for a field is less than one or exceeds the number of columns (or fields) in the current record. Similarly, an error occurs if a field with a name that is not one of the fieldnames of the table(s) that is(are) being queried.

**Example #4 - odbc_result() examples**

```
<?php
$item_3   = odbc_result($Query_ID, 3);
$item_val = odbc_result($Query_ID, "val");
?>
```

## Notes

Field indices start from 1. Regarding the way binary or long column data is returned refer

to odbc_binmode() and odbc_longreadlen().

# odbc_rollback

odbc_rollback -- Rollback a transaction

## Description

bool **odbc_rollback** ( resource `$connection_id` )

Rolls back all pending statements on the connection.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

## Return Values

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

# odbc_setoption

odbc_setoption -- Adjust ODBC settings

## Description

bool **odbc_setoption** ( resource `$id`, int `$function`, int `$option`, int `$param` )

This function allows fiddling with the ODBC options for a particular connection or query result. It was written to help find work around to problems in quirky ODBC drivers. You should probably only use this function if you are an ODBC programmer and understand the effects the various options will have. You will certainly need a good ODBC reference to explain all the different options and values that can be used. Different driver versions support different options.

Because the effects may vary depending on the ODBC driver, use of this function in scripts to be made publicly available is strongly discouraged. Also, some ODBC options are not available to this function because they must be set before the connection is established or the query is prepared. However, if on a particular job it can make PHP work so your boss doesn't tell you to use a commercial product, that's all that really matters.

## Parameters

*`id`*
: Is a connection id or result id on which to change the settings. For SQLSetConnectOption(), this is a connection id. For SQLSetStmtOption(), this is a result id.

*`function`*
: Is the ODBC function to use. The value should be 1 for SQLSetConnectOption() and 2 for SQLSetStmtOption().

*`option`*
: The option to set.

*`param`*
: The value for the given *`option`*.

## Return Values

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

## Examples

**Example #5 - odbc_setoption() examples**

```
<?php
// 1. Option 102 of SQLSetConnectOption() is SQL_AUTOCOMMIT.
//    Value 1 of SQL_AUTOCOMMIT is SQL_AUTOCOMMIT_ON.
//    This example has the same effect as
//    odbc_autocommit($conn, true);

odbc_setoption($conn, 1, 102, 1);

// 2. Option 0 of SQLSetStmtOption() is SQL_QUERY_TIMEOUT.
//    This example sets the query to timeout after 30 seconds.

$result = odbc_prepare($conn, $sql);
odbc_setoption($result, 2, 0, 30);
odbc_execute($result);
?>
```

# odbc_specialcolumns

odbc_specialcolumns -- Retrieves special columns

## Description

resource **odbc_specialcolumns** ( resource `$connection_id`, int `$type`, string `$qualifier` , string `$owner`, string `$table`, int `$scope`, int `$nullable` )

Retrieves either the optimal set of columns that uniquely identifies a row in the table, or columns that are automatically updated when any value in the row is updated by a transaction.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`type`*
When the type argument is **SQL_BEST_ROWID**, odbc_specialcolumns() returns the column or columns that uniquely identify each row in the table. When the type argument is **SQL_ROWVER**, odbc_specialcolumns() returns the column or columns in the specified table, if any, that are automatically updated by the data source when any value in the row is updated by any transaction.

*`qualifier`*
The qualifier.

*`owner`*
The owner.

*`table`*
The table.

*`scope`*
The scope, which orders the result set.

*`nullable`*
The nullable option.

## Return Values

Returns an ODBC result identifier or **FALSE** on failure.

The result set has the following columns:

- SCOPE

- COLUMN_NAME
- DATA_TYPE
- TYPE_NAME
- PRECISION
- LENGTH
- SCALE
- PSEUDO_COLUMN

# odbc_statistics

odbc_statistics -- Retrieve statistics about a table

## Description

resource **odbc_statistics** ( resource `$connection_id`, string `$qualifier`, string `$owner`, string `$table_name`, int `$unique`, int `$accuracy` )

Get statistics about a table and its indexes.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`qualifier`*
The qualifier.

*`owner`*
The owner.

*`table_name`*
The table name.

*`unique`*
The unique attribute.

*`accuracy`*
The accuracy.

## Return Values

Returns an ODBC result identifier or **FALSE** on failure.

The result set has the following columns:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- NON_UNIQUE
- INDEX_QUALIFIER
- INDEX_NAME
- TYPE

- SEQ_IN_INDEX
- COLUMN_NAME
- COLLATION
- CARDINALITY
- PAGES
- FILTER_CONDITION

The result set is ordered by NON_UNIQUE, TYPE, INDEX_QUALIFIER, INDEX_NAME and SEQ_IN_INDEX.

# odbc_tableprivileges

odbc_tableprivileges -- Lists tables and the privileges associated with each table

## Description

resource **odbc_tableprivileges** ( resource `$connection_id`, string `$qualifier`, string `$owner`, string `$name` )

Lists tables in the requested range and the privileges associated with each table.

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`qualifier`*
The qualifier.

*`owner`*
The owner. Accepts the following search patterns: ('%' to match zero or more characters and '_' to match a single character)

*`name`*
The name. Accepts the following search patterns: ('%' to match zero or more characters and '_' to match a single character)

## Return Values

An ODBC result identifier or **FALSE** on failure.

The result set has the following columns:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- GRANTOR
- GRANTEE
- PRIVILEGE
- IS_GRANTABLE

The result set is ordered by TABLE_QUALIFIER, TABLE_OWNER and TABLE_NAME.

# odbc_tables

odbc_tables -- Get the list of table names stored in a specific data source

## Description

resource **odbc_tables** ( resource `$connection_id` [, string `$qualifier` [, string `$owner` [, string `$name` [, string `$types` ] ] ] ] )

Lists all tables in the requested range.

To support enumeration of qualifiers, owners, and table types, the following special semantics for the *`qualifier`*, *`owner`*, *`name`*, and *`table_type`* are available:

- If *`qualifier`* is a single percent character (%) and *`owner`* and *`name`* are empty strings, then the result set contains a list of valid qualifiers for the data source. (All columns except the TABLE_QUALIFIER column contain NULLs.)
- If *`owner`* is a single percent character (%) and *`qualifier`* and *`name`* are empty strings, then the result set contains a list of valid owners for the data source. (All columns except the TABLE_OWNER column contain NULLs.)
- If *`table_type`* is a single percent character (%) and *`qualifier`*, *`owner`* and *`name`* are empty strings, then the result set contains a list of valid table types for the data source. (All columns except the TABLE_TYPE column contain NULLs.)

## Parameters

*`connection_id`*
The ODBC connection identifier, see odbc_connect() for details.

*`qualifier`*
The qualifier.

*`owner`*
The owner. Accepts search patterns ('%' to match zero or more characters and '_' to match a single character).

*`name`*
The name. Accepts search patterns ('%' to match zero or more characters and '_' to match a single character).

*`types`*
If *`table_type`* is not an empty string, it must contain a list of comma-separated values for the types of interest; each value may be enclosed in single quotes (') or unquoted. For example, "'TABLE','VIEW'" or "TABLE, VIEW". If the data source does not support a specified table type, odbc_tables() does not return any results for that type.

## Return Values

Returns an ODBC result identifier containing the information or **FALSE** on failure.

The result set has the following columns:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- TABLE_TYPE
- REMARKS

The result set is ordered by TABLE_TYPE, TABLE_QUALIFIER, TABLE_OWNER and TABLE_NAME.

## See Also

- odbc_tableprivileges()