Oracle OCI8

Introduction

These functions allow you to access Oracle 10, Oracle 9, Oracle 8 and Oracle 7 databases using the Oracle Call Interface (OCI). They support binding of PHP variables to Oracle placeholders, have full LOB, FILE and ROWID support, and allow you to use user-supplied define variables.

Installing/Configuring

Requirements

You will need the Oracle client libraries to use this extension. Windows users will need libraries with version at least 10 to use the *php_oci8.dll*.

Note
This extension does not support Oracle 8 client libraries anymore. Though you still can connect to Oracle 8 servers as long as the client library (v.9+) supports this.

The most convenient way to install all the required files is to use Oracle Instant Client, which is available from here: » http://www.oracle.com/technology/tech/oci/instantclient/instantclient.html. To work with OCI8 module "basic" version of Oracle Instant Client is enough. Instant Client does not need ORACLE_SID or ORACLE_HOME environment variables set. You still may need to set LD_LIBRARY_PATH and NLS_LANG, though.

Before using this extension, make sure that you have set up your Oracle environment variables properly for the Oracle user, as well as your web daemon user. These variables should be set up *before* you start your web-server. The variables you might need to set are as follows:

- ORACLE_HOME
- ORACLE_SID
- LD_PRELOAD
- LD_LIBRARY_PATH
- NLS_LANG

For less frequently used Oracle environment variables such as TNS_ADMIN, TWO_TASK, ORA_TZFILE, and the various Oracle globalization settings like ORA_NLS33, ORA_NLS10 and the NLS_* variables refer to Oracle documentation.

After setting up the environment variables for your web server user, be sure to also add the web server user (nobody, www) to the oracle group.

Note
If your web server doesn't start or crashes at startup Check that Apache is linked with the pthread library:

```
# ldd /www/apache/bin/httpd
  libpthread.so.0 => /lib/libpthread.so.0 (0x4001c000)
  libm.so.6 => /lib/libm.so.6 (0x4002f000)
  libcrypt.so.1 => /lib/libcrypt.so.1 (0x4004c000)
  libdl.so.2 => /lib/libdl.so.2 (0x4007a000)
  libc.so.6 => /lib/libc.so.6 (0x4007e000)
  /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

If the libpthread is not listed you have to reinstall Apache:

```
# cd /usr/src/apache_1.3.xx
# make clean
# LIBS=-lpthread ./config.status
# make
# make install
```

Please note that on some systems, like UnixWare it is libthread instead of libpthread. PHP and Apache have to be configured with EXTRA_LIBS=-lthread.

Installation

There is no installation needed to use these functions; they are part of the PHP core.

Runtime Configuration

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

OCI8 Configuration Options

Name	Default	Changeable	Changelog
oci8.privileged_connect	"0"	PHP_INI_SYSTEM	Available since PHP 5.1.2.
oci8.max_persistent	"-1"	PHP_INI_SYSTEM	Available since PHP 5.1.2.
oci8.persistent_timeout	"-1"	PHP_INI_SYSTEM	Available since PHP 5.1.2.
oci8.ping_interval	"60"	PHP_INI_SYSTEM	Available since PHP 5.1.2.

oci8.statement_cache_size	"20"	PHP_INI_SYSTEM	Available since PHP 5.1.2.
oci8.default_prefetch	"10"	PHP_INI_SYSTEM	Available since PHP 5.1.2.
oci8.old_oci_close_semantics	"0"	PHP_INI_SYSTEM	Available since PHP 5.1.2.

Here's a short explanation of the configuration directives.

oci8.privileged_connect boolean
: This option enables privileged connections using external credentials (**OCI_SYSOPER**, **OCI_SYSDBA**).

oci8.max_persistent int
: The maximum number of persistent OCI8 connections per process. Setting this option to -1 means that there is no limit.

oci8.persistent_timeout int
: The maximum length of time (in seconds) that a given process is allowed to maintain an idle persistent connection. Setting this option to -1 means that idle persistent connections will be maintained forever.

oci8.ping_interval int
: The length of time (in seconds) that must pass before issuing a ping during oci_pconnect(). When set to 0, persistent connections will be pinged every time they are reused. To disable pings completely, set this option to -1.

> **Note**
>
> Disabling pings will cause oci_pconnect() calls to operate at the highest efficiency, but may cause PHP to not detect faulty connections, such as those caused by network partitions, or if the Oracle server has gone down since PHP connected, until later in the script. Consult the oci_pconnect() documentation for more information.

oci8.statement_cache_size int
: This option enables statement caching, and specifies how many statements to cache. To disable statement caching just set this option to 0.

> **Note**
>
> A larger cache can result in improved performance, at the cost of increased memory usage.

oci8.default_prefetch int

This option enables statement prefetching and sets the default number of rows that will be fetched automatically after statement execution.

Note
A larger prefetch can result in improved performance, at the cost of increased memory usage.

oci8.old_oci_close_semantics boolean

This option controls oci_close() behaviour. Enabling it means that oci_close() will do nothing; the connection will not be closed until the end of the script. This is for backward compatibility only. If you find that you need to enable this setting, you are *strongly encouraged* to remove the oci_close() calls from your application instead of enabling this option.

Resource Types

This extension has no resource types defined.

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.

OCI_DEFAULT (integer)
Statement execution mode. Statement is not committed automatically when using this mode.

OCI_DESCRIBE_ONLY (integer)
Statement execution mode. Use this mode if you don't want to execute the query, but get the select-list's description.

OCI_COMMIT_ON_SUCCESS (integer)
Statement execution mode. Statement is automatically committed after oci_execute() call.

OCI_EXACT_FETCH (integer)
Statement fetch mode. Used when the application knows in advance exactly how many rows it will be fetching. This mode turns prefetching off for Oracle release 8 or later mode. Cursor is cancelled after the desired rows are fetched and may result in reduced server-side resource usage.

OCI_SYSDATE (integer)

OCI_B_BFILE (integer)
Used with oci_bind_by_name() when binding BFILEs.

OCI_B_CFILEE (integer)
Used with oci_bind_by_name() when binding CFILEs.

OCI_B_CLOB (integer)
Used with oci_bind_by_name() when binding CLOBs.

OCI_B_BLOB (integer)
Used with oci_bind_by_name() when binding BLOBs.

OCI_B_ROWID (integer)
Used with oci_bind_by_name() when binding ROWIDs.

OCI_B_CURSOR (integer)
Used with oci_bind_by_name() when binding cursors, previously allocated with oci_new_descriptor().

OCI_B_NTY (integer)
Used with oci_bind_by_name() when binding named data types. Note: in PHP < 5.0 it was called **OCI_B_SQLT_NTY**.

OCI_B_BIN (integer)

SQLT_BFILEE (integer)
The same as **OCI_B_BFILE**.

SQLT_CFILEE (integer)
The same as **OCI_B_CFILEE**.

SQLT_CLOB (integer)
The same as **OCI_B_CLOB**.

SQLT_BLOB (integer)
The same as **OCI_B_BLOB**.

SQLT_RDD (integer)
The same as **OCI_B_ROWID**.

SQLT_NTY (integer)
The same as **OCI_B_NTY**.

SQLT_LNG (integer)
Used with oci_bind_by_name() to bind LONG values.

SQLT_LBI (integer)
Used with oci_bind_by_name() to bind LONG RAW values.

SQLT_BIN (integer)
Used with oci_bind_by_name() to bind RAW values.

SQLT_NUM (integer)
Used with oci_bind_array_by_name() to bind arrays of NUMBER.

SQLT_INT (integer)
Used with oci_bind_array_by_name() to bind arrays of INTEGER.

SQLT_AFC (integer)
Used with oci_bind_array_by_name() to bind arrays of CHAR.

SQLT_CHR (integer)
Used with oci_bind_array_by_name() to bind arrays of VARCHAR2. Also used with oci_bind_by_name().

SQLT_VCS (integer)
Used with oci_bind_array_by_name() to bind arrays of VARCHAR.

SQLT_AVC (integer)
Used with oci_bind_array_by_name() to bind arrays of CHARZ.

SQLT_STR (integer)
Used with oci_bind_array_by_name() to bind arrays of STRING.

SQLT_LVC (integer)
Used with oci_bind_array_by_name() to bind arrays of LONG VARCHAR.

SQLT_FLT (integer)
Used with oci_bind_array_by_name() to bind arrays of FLOAT.

SQLT_ODT (integer)
Used with oci_bind_array_by_name() to bind arrays of LONG.

SQLT_BDOUBLE (integer)

SQLT_BFLOAT (integer)

OCI_FETCHSTATEMENT_BY_COLUMN (integer)
Default mode of oci_fetch_all().

OCI_FETCHSTATEMENT_BY_ROW (integer)
Alternative mode of oci_fetch_all().

OCI_ASSOC (integer)
Used with oci_fetch_all() and oci_fetch_array() to get an associative array as a result.

OCI_NUM (integer)
Used with oci_fetch_all() and oci_fetch_array() to get an enumerated array as a result.

OCI_BOTH (integer)
Used with oci_fetch_all() and oci_fetch_array() to get an array with both associative and number indices.

OCI_RETURN_NULLS (integer)
Used with oci_fetch_array() to get empty array elements if field's value is **NULL**.

OCI_RETURN_LOBS (integer)
Used with oci_fetch_array() to get value of LOB instead of the descriptor.

OCI_DTYPE_FILE (integer)
This flag tells oci_new_descriptor() to initialize new FILE descriptor.

OCI_DTYPE_LOB (integer)
This flag tells oci_new_descriptor() to initialize new LOB descriptor.

OCI_DTYPE_ROWID (integer)
This flag tells oci_new_descriptor() to initialize new ROWID descriptor.

OCI_D_FILE (integer)
The same as **OCI_DTYPE_FILE**.

OCI_D_LOB (integer)
The same as **OCI_DTYPE_LOB**.

OCI_D_ROWID (integer)
The same as **OCI_DTYPE_ROWID**.

OCI_SYSOPER (integer)
Used with oci_connect() to connect as SYSOPER using external credentials (oci8.privileged_connect should be enabled for this).

OCI_SYSDBA (integer)
Used with oci_connect() to connect as SYSDBA using external credentials (oci8.privileged_connect should be enabled for this).

OCI_LOB_BUFFER_FREE (integer)
Used with OCI-Lob->flush to free buffers used.

OCI_TEMP_CLOB (integer)
Used with OCI-Lob->writeTemporary to indicate explicilty that temporary CLOB should be created.

OCI_TEMP_BLOB (integer)
Used with OCI-Lob->writeTemporary to indicate explicilty that temporary BLOB should be created.

Examples

Example #1 - Basic query

```
<?php

 $conn = oci_connect('hr', 'hr', 'orcl');
 if (!$conn) {
   $e = oci_error();
   print htmlentities($e['message']);
   exit;
 }

 $query = 'SELECT * FROM DEPARTMENTS';

 $stid = oci_parse($conn, $query);
 if (!$stid) {
   $e = oci_error($conn);
   print htmlentities($e['message']);
   exit;
 }

 $r = oci_execute($stid, OCI_DEFAULT);
 if (!$r) {
   $e = oci_error($stid);
   echo htmlentities($e['message']);
   exit;
 }

 print '<table border="1">';
 while ($row = oci_fetch_array($stid, OCI_RETURN_NULLS)) {
   print '<tr>';
      foreach ($row as $item) {
        print '<td>'.($item?htmlentities($item):' ').'</td>';
      }
      print '</tr>';
 }
 print '</table>';

 oci_close($conn);
?>
```

Example #2 - Insert with bind variables

```
<?php

 // Before running, create the table:
 //   CREATE TABLE MYTABLE (mid NUMBER, myd VARCHAR2(20));
```

```
 $conn = oci_connect('scott', 'tiger', 'orcl');

 $query = 'INSERT INTO MYTABLE VALUES(:myid, :mydata)';

 $stid = oci_parse($conn, $query);

 $id = 60;
 $data = 'Some data';

 oci_bind_by_name($stid, ':myid', $id);
 oci_bind_by_name($stid, ':mydata', $data);

 $r = oci_execute($stid);

 if ($r)
   print "One row inserted";

 oci_close($conn);

?>
```

Example #3 - Inserting data into a CLOB column

```
<?php

// Before running, create the table:
//     CREATE TABLE MYTABLE (mykey NUMBER, myclob CLOB);

$conn = oci_connect('scott', 'tiger', 'orcl');

$mykey = 12343;  // arbitrary key for this example;

$sql = "INSERT INTO mytable (mykey, myclob)
       VALUES (:mykey, EMPTY_CLOB())
       RETURNING myclob INTO :myclob";

$stid = oci_parse($conn, $sql);
$clob = oci_new_descriptor($conn, OCI_D_LOB);
oci_bind_by_name($stid, ":mykey", $mykey, 5);
oci_bind_by_name($stid, ":myclob", $clob, -1, OCI_B_CLOB);
oci_execute($stid, OCI_DEFAULT);
$clob->save("A very long string");

oci_commit($conn);

// Fetching CLOB data

$query = 'SELECT myclob FROM mytable WHERE mykey = :mykey';

$stid = oci_parse ($conn, $query);
oci_bind_by_name($stid, ":mykey", $mykey, 5);
oci_execute($stid, OCI_DEFAULT);

print '<table border="1">';
while ($row = oci_fetch_array($stid, OCI_ASSOC)) {
```

```
 $result = $row['MYCLOB']->load();
 print '<tr><td>'.$result.'</td></tr>';
}
print '</table>';

?>
```

You can easily access stored procedures in the same way as you would from the command line.

Example #4 - Using Stored Procedures

```
<?php
// by webmaster at remoterealty dot com
$sth = oci_parse($dbh, "begin sp_newaddress( :address_id, '$firstname',
'$lastname', '$company', '$address1', '$address2', '$city', '$state',
'$postalcode', '$country', :error_code );end;");

// This calls stored procedure sp_newaddress, with :address_id being an
// in/out variable and :error_code being an out variable.
// Then you do the binding:

  oci_bind_by_name($sth, ":address_id", $addr_id, 10);
  oci_bind_by_name($sth, ":error_code", $errorcode, 10);
  oci_execute($sth);

?>
```

Connecting Handling

The oci8 extension provides you with 3 different functions for connecting to Oracle. It is up to you to use the most appropriate function for your application, and the information in this section is intended to help you make an informed choice.

Connecting to an Oracle server is a reasonably expensive operation, in terms of the time that it takes to complete. The oci_pconnect() function uses a persistent cache of connections that can be re-used across different script requests. This means that you will typically only incur the connection overhead once per php process (or apache child).

If your application connects to Oracle using a different set of credentials for each web user, the persistent cache employed by oci_pconnect() will become less useful as the number of concurrent users increases, to the point where it may start to adversely affect the overall performance of your Oracle server due to maintaining too many idle connections. If your application is structured in this way, it is recommended that you either tune your application using the oci8.max_persistent and oci8.persistent_timeout configuration settings (these will give you control over the persistent connection cache size and lifetime) or use oci_connect() instead.

Both oci_connect() and oci_pconnect() employ a connection cache; if you make multiple calls to oci_connect(), using the same parameters, in a given script, the second and subsequent calls return the existing connection handle. The cache used by oci_connect() is cleaned up at the end of the script run, or when you explicitly close the connection handle. oci_pconnect() has similar behaviour, although its cache is maintained separately and survives between requests.

This caching feature is important to remember, because it gives the appearance that the two handles are not transactionally isolated (they are in fact the same connection handle, so there is no isolation of any kind). If your application needs two separate, transactionally isolated connections, you should use oci_new_connect().

oci_new_connect() always creates a new connection to the Oracle server, regardless of what other connections might already exist. High traffic web applications should try to avoid using oci_new_connect(), especially in the busiest sections of the application.

Supported Datatypes

The driver supports the following types when binding parameters using oci_bind_by_name() function:

Type	Mapping
SQLT_NTY	Maps a native collection type from a PHP collection object, such as those created by oci_new_collection().
SQLT_BFILEE	Maps a native descriptor, such as those created by oci_new_descriptor().
SQLT_CFILEE	Maps a native descriptor, such as those created by oci_new_descriptor().
SQLT_CLOB	Maps a native descriptor, such as those created by oci_new_descriptor().
SQLT_BLOB	Maps a native descriptor, such as those created by oci_new_descriptor().
SQLT_RDD	Maps a native descriptor, such as those created by oci_new_descriptor().
SQLT_NUM	Converts the PHP parameter to a 'C' long type, and binds to that value.
SQLT_RSET	Maps a native statement handle, such as those created by oci_parse() or those retrieved from other OCI queries.
SQLT_CHR and any other type	Converts the PHP parameter to a string type and binds as a string.

The following types are supported when retrieving columns from a result set:

Type	Mapping
SQLT_RSET	Creates an oci statement resource to represent the the cursor.
SQLT_RDD	Creates a ROWID object.
SQLT_BLOB	Creates a LOB object.

SQLT_CLOB	Creates a LOB object.
SQLT_BFILE	Creates a LOB object.
SQLT_LNG	Bound as SQLT_CHR, returned as a string
SQLT_LBI	Bound as SQLT_BIN, returned as a string
Any other type	Bound as SQLT_CHR, returned as a string

OCI8 Functions

oci_bind_array_by_name

oci_bind_array_by_name -- Binds PHP array to Oracle PL/SQL array by name

Description

bool **oci_bind_array_by_name** (resource `$statement`, string `$name`, array `&$var_array`, int `$max_table_length` [, int `$max_item_length` [, int `$type`]])

Binds the PHP array *`var_array`* to the Oracle placeholder *`name`*, which points to Oracle PL/SQL array. Whether it will be used for input or output will be determined at run-time.

Parameters

`statement`
A valid OCI statement identifier.

`name`
The Oracle placeholder.

`var_array`
An array.

`max_table_length`
Sets the maximum length both for incoming and result arrays.

`max_item_length`
Sets maximum length for array items. If not specified or equals to -1, oci_bind_array_by_name() will use find the longest element in the incoming array and will use it as maximum length for array items.

`type`
Should be used to set the type of PL/SQL array items. See list of available types below:

- **SQLT_NUM** - for arrays of NUMBER.
- **SQLT_INT** - for arrays of INTEGER (Note: INTEGER it is actually a synonym for NUMBER(38), but **SQLT_NUM** type won't work in this case even though they are synonyms).
- **SQLT_FLT** - for arrays of FLOAT.
- **SQLT_AFC** - for arrays of CHAR.
- **SQLT_CHR** - for arrays of VARCHAR2.
- **SQLT_VCS** - for arrays of VARCHAR.
- **SQLT_AVC** - for arrays of CHARZ.

- **SQLT_STR** - for arrays of STRING.
- **SQLT_LVC** - for arrays of LONG VARCHAR.
- **SQLT_ODT** - for arrays of DATE.

Return Values

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

Examples

Example #5 - oci_bind_array_by_name() **example**

```
<?php

$c = oci_connect("scott", "tiger");

$create = "CREATE TABLE bind_example(name VARCHAR(20))";
$statement = oci_parse($c, $create);
oci_execute($statement);

$create_pkg = "
CREATE OR REPLACE PACKAGE ARRAYBINDPKG1 AS
 TYPE ARRTYPE IS TABLE OF VARCHAR(20) INDEX BY BINARY_INTEGER;
 PROCEDURE iobind(c1 IN OUT ARRTYPE);
END ARRAYBINDPKG1;";
$statement = oci_parse($c, $create_pkg);
oci_execute($statement);

$create_pkg_body = "
CREATE OR REPLACE PACKAGE BODY ARRAYBINDPKG1 AS
 CURSOR CUR IS SELECT name FROM bind_example;
 PROCEDURE iobind(c1 IN OUT ARRTYPE) IS
   BEGIN
   FOR i IN 1..5 LOOP
     INSERT INTO bind_example VALUES (c1(i));
   END LOOP;
   IF NOT CUR%ISOPEN THEN
     OPEN CUR;
   END IF;
   FOR i IN REVERSE 1..5 LOOP
     FETCH CUR INTO c1(i);
     IF CUR%NOTFOUND THEN
       CLOSE CUR;
       EXIT;
     END IF;
   END LOOP;
 END iobind;
END ARRAYBINDPKG1;";
$statement = oci_parse($c, $create_pkg_body);
oci_execute($statement);
```

```
$statement = oci_parse($c, "BEGIN ARRAYBINDPKG1.iobind(:c1); END;");
$array = array("one", "two", "three", "four", "five");
oci_bind_array_by_name($statement, ":c1", $array, 5, -1, SQLT_CHR);
oci_execute($statement);
var_dump($array);
?>
```

Notes

Note
This function is available since OCI8 release 1.2.

oci_bind_by_name

oci_bind_by_name -- Binds the PHP variable to the Oracle placeholder

Description

bool **oci_bind_by_name** (resource `$statement`, string `$ph_name`, mixed `&$variable` [, int `$maxlength` [, int `$type`]])

Binds the PHP variable *`variable`* to the Oracle placeholder *`ph_name`*. Whether it will be used for input or output will be determined at run-time and the necessary storage space will be allocated.

Parameters

`statement`
An OCI statement.

`ph_name`
The placeholder.

`variable`
The PHP variable.

`maxlength`
Sets the maximum length for the bind. If you set it to -1, this function will use the current length of *`variable`* to set the maximum length.

`type`
If you need to bind an abstract datatype (LOB/ROWID/BFILE) you need to allocate it first using the oci_new_descriptor() function. The *`length`* is not used for abstract datatypes and should be set to -1. The *`type`* parameter tells Oracle which descriptor is used. Default to **SQLT_CHR**. Possible values are:

- **SQLT_FILE** - for BFILEs;
- **SQLT_CFILE** - for CFILEs;
- **SQLT_CLOB** - for CLOBs;
- **SQLT_BLOB** - for BLOBs;
- **SQLT_RDD** - for ROWIDs;
- **SQLT_NTY** - for named datatypes;
- **SQLT_INT** - for integers;
- **SQLT_CHR** - for VARCHARs;
- **SQLT_BIN** - for RAW columns;
- **SQLT_LNG** - for LONG columns;

- **SQLT_LBI** - for LONG RAW columns;
- **SQLT_RSET** - for cursors, that were created before with oci_new_cursor().

Return Values

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

Examples

Example #6 - oci_bind_by_name() example

```
<?php
/* oci_bind_by_name example thies at thieso dot net (980221)
 inserts 3 records into emp, and uses the ROWID for updating the
 records just after the insert.
*/

$conn = oci_connect("scott", "tiger");

$stmt = oci_parse($conn, "
                         INSERT INTO
                                   emp (empno, ename)
                              VALUES
                                   (:empno,:ename)
                           RETURNING
                                   ROWID
                                INTO
                                   :rid
                        ");

$data = array(
             1111 => "Larry",
             2222 => "Bill",
             3333 => "Jim"
            );

$rowid = oci_new_descriptor($conn, OCI_D_ROWID);

oci_bind_by_name($stmt, ":empno", $empno, 32);
oci_bind_by_name($stmt, ":ename", $ename, 32);
oci_bind_by_name($stmt, ":rid",   $rowid, -1, OCI_B_ROWID);

$update = oci_parse($conn, "
                           UPDATE
                                 emp
                              SET
                                 sal = :sal
                            WHERE
                                 ROWID = :rid
                          ");
oci_bind_by_name($update, ":rid", $rowid, -1, OCI_B_ROWID);
oci_bind_by_name($update, ":sal", $sal,   32);
```

```
$sal = 10000;

foreach ($data as $empno => $ename) {
   oci_execute($stmt);
   oci_execute($update);
}

$rowid->free();

oci_free_statement($update);
oci_free_statement($stmt);

$stmt = oci_parse($conn, "
                          SELECT
                                *
                            FROM
                                emp
                           WHERE
                                empno
                              IN
                                (1111,2222,3333)
                         ");
oci_execute($stmt);

while ($row = oci_fetch_assoc($stmt)) {
   var_dump($row);
}

oci_free_statement($stmt);

/* delete our "junk" from the emp table.... */
$stmt = oci_parse($conn, "
                          DELETE FROM
                                     emp
                                WHERE
                                     empno
                                   IN
                                     (1111,2222,3333)
                         ");
oci_execute($stmt);
oci_free_statement($stmt);

oci_close($conn);
?>
```

Remember, this function strips trailing whitespaces. See the following example:

Example #7 - oci_bind_by_name() **example**

```
<?php
   $connection = oci_connect('apelsin','kanistra');
   $query = "INSERT INTO test_table VALUES(:id, :text)";

   $statement = oci_parse($query);
```

```
   oci_bind_by_name($statement, ":id", 1);
   oci_bind_by_name($statement, ":text", "trailing spaces follow     ");
   oci_execute($statement);
   /*
    This code will insert into DB string 'trailing spaces follow', without
    trailing spaces
   */
?>
```

Example #8 - oci_bind_by_name() example

```
<?php
   $connection = oci_connect('apelsin','kanistra');
   $query = "INSERT INTO test_table VALUES(:id, 'trailing spaces follow
')";

   $statement = oci_parse($query);
   oci_bind_by_name($statement, ":id", 1);
   oci_execute($statement);
   /*
    And this code will add 'trailing spaces follow      ', preserving
    trailing whitespaces
   */
?>
```

Return Values

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

Notes

Warning

Do not use magic_quotes_gpc or addslashes() and oci_bind_by_name() simultaneously as no quoting is needed and any magically applied quotes will be written into your database as oci_bind_by_name() is not able to distinguish magically added quotings from those added intentionally.

Note

In PHP versions before 5.0.0 you must use ocibindbyname() instead. This name still can be used, it was left as alias of oci_bind_by_name() for downwards compatability. This, however, is deprecated and not recommended.

oci_cancel

oci_cancel -- Cancels reading from cursor

Description

bool **oci_cancel** (resource `$statement`)

Invalidates a cursor, freeing all associated resources and cancels the ability to read from it.

Parameters

`statement`
An OCI statement.

Return Values

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

oci_close

oci_close -- Closes Oracle connection

Description

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

Closes the Oracle *`connection`*.

Parameters

`connection`
An Oracle connection identifier, returned by oci_connect().

Return Values

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

Notes

Note
Starting from the version 1.1 oci_close() correctly closes the Oracle connection. Use oci8.old_oci_close_semantics option to restore old behaviour of this function.

OCI-Collection->append

OCI-Collection->append -- Appends element to the collection

Description

OCI-Collection

bool **append** (mixed `$value`)

Appends element to the end of the collection.

Parameters

`value`
: The value to be added to the collection. Can be a string or a number.

Return Values

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

See Also

- OCI-Collection->assign

OCI-Collection->assign

OCI-Collection->assign -- Assigns a value to the collection from another existing collection

Description

OCI-Collection

bool **assign** (OCI-Collection `$from`)

Assigns a value to the collection from another, previously created collection. Both collections must be created with oci_new_collection() prior to using them.

Parameters

`from`
An instance of OCI-Collection.

Return Values

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

See Also

- OCI-Collection->append

OCI-Collection->assignElem

OCI-Collection->assignElem -- Assigns a value to the element of the collection

Description

OCI-Collection

bool **assignElem** (int `$index`, mixed `$value`)

Assigns a value to the element with index *`index`*.

Parameters

`index`
: The element index. First index is 1.

`value`
: Can be a string or a number.

Return Values

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

See Also

- OCI-Collection->getElem

OCI-Collection->free

OCI-Collection->free -- Frees the resources associated with the collection object

Description

OCI-Collection

bool **free** (void)

Frees the resources associated with the collection object.

Return Values

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

See Also

- oci_new_collection

OCI-Collection->getElem

OCI-Collection->getElem -- Returns value of the element

Description

OCI-Collection

mixed **getElem** (int `$index`)

Returns element's value with the index *`index`* (1-based).

Parameters

`index`
: The element index. First index is 1.

Return Values

Returns **FALSE** if such element doesn't exist; **NULL** if element is **NULL**; string if element is column of a string datatype or number if element is numeric field.

See Also

- OCI-Collection->assignElem

OCI-Collection->max

OCI-Collection->max -- Returns the maximum number of elements in the collection

Description

OCI-Collection

int **max** (void)

Returns the maximum number of elements in the collection.

Return Values

Returns the maximum number as an integer, or **FALSE** on errors.

If the returned value is 0, then the number of elements is not limited.

See Also

- OCI-Collection->size

OCI-Collection->size

OCI-Collection->size -- Returns size of the collection

Description

OCI-Collection

int **size** (void)

Returns the size of the collection.

Return Values

Returns the number of elements in the collection or **FALSE** on error.

See Also

- OCI-Collection->max

OCI-Collection->trim

OCI-Collection->trim -- Trims elements from the end of the collection

Description

OCI-Collection

bool **trim** (int `$num`)

Trims *`num`* of elements from the end of the collection.

Parameters

`num`
: The number of elements to be trimmed.

Return Values

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

See Also

- OCI-Collection->size

oci_commit

oci_commit -- Commits outstanding statements

Description

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

Commits all outstanding statements for the active transaction on the Oracle *`connection`*.

Parameters

`connection`
An Oracle connection identifier, returned by oci_connect() or oci_pconnect().

Return Values

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

Examples

Example #9 - oci_commit() example

```
<?php
// Login to Oracle server
$conn = oci_connect('scott', 'tiger');

// Parse SQL
$stmt = oci_parse($conn, "
                         INSERT INTO
                                   employees (name, surname)
                              VALUES
                                   ('Maxim', 'Maletsky')
                        ");

/* Execute statement
  OCI_DEFAULT tells oci_execute()
  not to commit statement immediately */
oci_execute($stmt, OCI_DEFAULT);

/*
....
Parsing and executing other statements here ...
....
*/

// Commit transaction
$committed = oci_commit($conn);
```

```
// Test whether commit was successful. If error occurred, return error
message
if (!$committed) {
   $error = oci_error($conn);
   echo 'Commit failed. Oracle reports: ' . $error['message'];
}

?>
```

Notes

Note
Transactions are automatically rolled back when you close the connection, or when the script ends, whichever is soonest. You need to explicitly call oci_commit() to commit the transaction, or oci_rollback() to abort it.

See Also

- oci_rollback()
- oci_execute()

oci_connect

oci_connect -- Establishes a connection to the Oracle server

Description

resource **oci_connect** (string `$username`, string `$password` [, string `$db` [, string `$charset` [, int `$session_mode`]]])

Returns a connection identifier needed for most other OCI calls.

Parameters

`username`
The Oracle user name.

`password`
The password for *`username`*.

`db`
This optional parameter can either contain the name of the local Oracle instance or the name of the entry in *tnsnames.ora*. If the not specified, PHP uses environment variables *ORACLE_SID* and *TWO_TASK* to determine the name of local Oracle instance and location of *tnsnames.ora* accordingly.

`charset`
Using Oracle server version 9.2 and greater, you can indicate *`charset`* by parameter, which will be used in the new connection. If you're using Oracle server < 9.2, this parameter will be ignored and the *NLS_LANG* environment variable will be used instead.

`session_mode`
This parameter is available since version 1.1 and accepts the following values: **OCI_DEFAULT**, **OCI_SYSOPER** and **OCI_SYSDBA**. If either **OCI_SYSOPER** or **OCI_SYSDBA** were specified, this function will try to establish privileged connection using external credentials. Privileged connections are disabled by default. To enable them you need to set oci8.privileged_connect to *On*.

Return Values

Returns a connection identifier or **FALSE** on error.

Examples

Example #10 - oci_connect() example

```
<?php
echo "<pre>";
$db = "";

$c1 = oci_connect("scott", "tiger", $db);
$c2 = oci_connect("scott", "tiger", $db);

function create_table($conn)
{
 $stmt = oci_parse($conn, "create table scott.hallo (test varchar2(64))");
 oci_execute($stmt);
 echo $conn . " created table\n\n";
}

function drop_table($conn)
{
 $stmt = oci_parse($conn, "drop table scott.hallo");
 oci_execute($stmt);
 echo $conn . " dropped table\n\n";
}

function insert_data($conn)
{
 $stmt = oci_parse($conn, "insert into scott.hallo
           values('$conn' || ' ' || to_char(sysdate,'DD-MON-YY
HH24:MI:SS'))");
 oci_execute($stmt, OCI_DEFAULT);
 echo $conn . " inserted hallo\n\n";
}

function delete_data($conn)
{
 $stmt = oci_parse($conn, "delete from scott.hallo");
 oci_execute($stmt, OCI_DEFAULT);
 echo $conn . " deleted hallo\n\n";
}

function commit($conn)
{
 oci_commit($conn);
 echo $conn . " committed\n\n";
}

function rollback($conn)
{
 oci_rollback($conn);
 echo $conn . " rollback\n\n";
}

function select_data($conn)
{
 $stmt = oci_parse($conn, "select * from scott.hallo");
 oci_execute($stmt, OCI_DEFAULT);
 echo $conn."----selecting\n\n";
 while (oci_fetch($stmt)) {
   echo $conn . " [" . oci_result($stmt, "TEST") . "]\n\n";
 }
```

```
 echo $conn . "----done\n\n";
}

create_table($c1);
insert_data($c1);   // Insert a row using c1
insert_data($c2);   // Insert a row using c2

select_data($c1);   // Results of both inserts are returned
select_data($c2);

rollback($c1);      // Rollback using c1

select_data($c1);   // Both inserts have been rolled back
select_data($c2);

insert_data($c2);   // Insert a row using c2
commit($c2);        // Commit using c2

select_data($c1);   // Result of c2 insert is returned

delete_data($c1);   // Delete all rows in table using c1
select_data($c1);   // No rows returned
select_data($c2);   // No rows returned
commit($c1);        // Commit using c1

select_data($c1);   // No rows returned
select_data($c2);   // No rows returned

drop_table($c1);
echo "</pre>";
?>
```

Notes

Note

If you're using PHP with Oracle Instant Client, you can use easy connect naming method described here: » http://download-west.oracle.com/docs/cd/B12037_01/network.101/b10775/naming.htm#i498306. Basically this means you can specify "//db_host[:port]/database_name" as database name. But if you want to use the old way of naming you *must* set either **ORACLE_HOME** or **TNS_ADMIN**.

Note

The second and subsequent calls to oci_connect() with the same parameters will return the connection handle returned from the first call. This means that queries issued against one handle are also applied to the other handles, because they are the *same* handle. This behaviour is demonstrated in Example 1 below. If you require two handles to be transactionally isolated from each other, you should use oci_new_connect() instead.

Note
In PHP versions before 5.0.0 you must use ocilogon() instead. This name still can be used, it was left as the alias of oci_connect() for downwards compatability. This, however, is deprecated and not recommended.

See Also

- oci_pconnect()
- oci_new_connect()
- oci_close()

oci_define_by_name

oci_define_by_name -- Uses a PHP variable for the define-step during a SELECT

Description

bool **oci_define_by_name** (resource `$statement`, string `$column_name`, mixed `&$variable` [, int `$type`])

Defines PHP variables for fetches of SQL-Columns.

Parameters

`statement`
A valid OCI statement identifier.

`column_name`
The column name. Must be uppercased. Take into consideration that Oracle uses ALL-UPPERCASE column names, whereby in your select you can also use lowercase. If you define a variable that doesn't exists in your select statement, no error will be issued.

`variable`
The PHP variable.

`type`
If you need to define an abstract datatype (LOB/ROWID/BFILE) you must allocate it first using oci_new_descriptor(). See also the oci_bind_by_name() function.

Return Values

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

Examples

Example #11 - oci_define_by_name() example

```
<?php
/* oci_define_by_name example - thies at thieso dot net (980219) */

$conn = oci_connect("scott", "tiger");

$stmt = oci_parse($conn, "SELECT empno, ename FROM emp");

/* the define MUST be done BEFORE oci_execute! */
```

```
oci_define_by_name($stmt, "EMPNO", $empno);
oci_define_by_name($stmt, "ENAME", $ename);

oci_execute($stmt);

while (oci_fetch($stmt)) {
   echo "empno:" . $empno . "\n";
   echo "ename:" . $ename . "\n";
}

oci_free_statement($stmt);
oci_close($conn);
?>
```

Notes

Note
In PHP versions before 5.0.0 you must use ocidefinebyname() instead. This name still can be used, it was left as alias of oci_define_by_name() for downwards compatability. This, however, is deprecated and not recommended.

oci_error

oci_error -- Returns the last error found

Description

array **oci_error** ([resource `$source`])

Returns the last error found.

Parameters

`source`
: For most errors, the parameter is the most appropriate resource handle. For connection errors with oci_connect(), oci_new_connect() or oci_pconnect() do not pass a parameter.

Return Values

If no error is found, oci_error() returns **FALSE**. oci_error() returns the error as an associative array. In this array, *code* consists the oracle error code and *message* the oracle error string.

ChangeLog

Version	Description
4.3	*offset* and *sqltext* will also be included in the return array to indicate the location of the error and the original SQL text which caused it.

Examples

Example #12 - Displaying the Oracle error message after a connection error

```
$conn = @oci_connect("scott", "tiger", "mydb");
if (!$conn) {
 $e = oci_error();   // For oci_connect errors pass no handle
 echo htmlentities($e['message']);
}
```

Example #13 - Displaying the Oracle error message after a parsing error

```
$stmt = @oci_parse($conn, "select ' from dual");  // note mismatched quote
if (!$stmt) {
 $e = oci_error($conn);  // For oci_parse errors pass the connection handle
 echo htmlentities($e['message']);
}
```

Example #14 - Displaying the Oracle error message and problematic statement after an execution error

```
$r = oci_execute($stmt);
if (!$r) {
 $e = oci_error($stmt); // For oci_execute errors pass the statementhandle
 echo htmlentities($e['message']);
 echo "<pre>";
 echo htmlentities($e['sqltext']);
 printf("\n%".($e['offset']+1)."s", "^");
 echo "</pre>";
}
```

Notes

Note

In PHP versions before 5.0.0 you must use ocierror() instead. This name still can be used, it was left as alias of oci_error() for downwards compatability. This, however, is deprecated and not recommended.

oci_execute

oci_execute -- Executes a statement

Description

bool **oci_execute** (resource `$statement` [, int `$mode`])

Executes a previously parsed *`statement`*.

Parameters

`statement`
A valid OCI statement identifier.

`mode`
Allows you to specify the execution mode (default is **OCI_COMMIT_ON_SUCCESS**). If you don't want statements to be committed automatically, you should specify **OCI_DEFAULT** as your *`mode`*. When using **OCI_DEFAULT** mode, you're creating a transaction. Transactions are automatically rolled back when you close the connection, or when the script ends, whichever is soonest. You need to explicitly call oci_commit() to commit the transaction, or oci_rollback() to abort it.

Return Values

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

Notes

Note
In PHP versions before 5.0.0 you must use ociexecute() instead. This name still can be used, it was left as alias of oci_execute() for downwards compatability. This, however, is deprecated and not recommended.

See Also

- oci_parse()

oci_fetch_all

oci_fetch_all -- Fetches all rows of result data into an array

Description

int **oci_fetch_all** (resource `$statement`, array `&$output` [, int `$skip` [, int `$maxrows` [, int `$flags`]]])

Fetches all the rows from a result into a user-defined array.

For details on the data type mapping performed by the oci8 driver, see the datatypes supported by the driver

Parameters

`statement`
A valid OCI statement identifier.

`output`

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

`skip`
The number of initial rows to ignore when fetching the result (default value of 0, to start at the first line).

`maxrows`
The number of rows to read, starting at the *`skip`* th row (default to -1, meaning all the rows).

`flags`
Parameter *`flags`* can be any combination of the following:

- **OCI_FETCHSTATEMENT_BY_ROW**
- **OCI_FETCHSTATEMENT_BY_COLUMN** (default value)
- **OCI_NUM**
- **OCI_ASSOC**

Return Values

Returns the number of rows fetched or **FALSE** in case of an error.

Examples

Example #15 - oci_fetch_all() example

```
<?php
/* oci_fetch_all example mbritton at verinet dot com (990624) */

$conn = oci_connect("scott", "tiger");

$stmt = oci_parse($conn, "select * from emp");

oci_execute($stmt);

$nrows = oci_fetch_all($stmt, $results);
if ($nrows > 0) {
  echo "<table border=\"1\">\n";
  echo "<tr>\n";
  foreach ($results as $key => $val) {
     echo "<th>$key</th>\n";
  }
  echo "</tr>\n";

  for ($i = 0; $i < $nrows; $i++) {
     echo "<tr>\n";
     foreach ($results as $data) {
        echo "<td>$data[$i]</td>\n";
     }
     echo "</tr>\n";
  }
  echo "</table>\n";
} else {
  echo "No data found<br />\n";
}
echo "$nrows Records Selected<br />\n";

oci_free_statement($stmt);
oci_close($conn);
?>
```

Notes

Note

In PHP versions before 5.0.0 you must use ocifetchstatement() instead. This name still can be used, it was left as alias of oci_fetch_all() for downwards compatability. This, however, is deprecated and not recommended.

oci_fetch_array

oci_fetch_array -- Returns the next row from the result data as an associative or numeric array, or both

Description

array **oci_fetch_array** (resource `$statement` [, int `$mode`])

Returns an array, which corresponds to the next result row.

For details on the data type mapping performed by the oci8 driver, see the datatypes supported by the driver

It should be mentioned here, that oci_fetch_array() is *insignificantly* slower, than oci_fetch_row(), but much more handy.

Parameters

`statement`
A valid OCI statement identifier.

`statement`
An optional second parameter can be any combination of the following constants:

- **OCI_BOTH** - return an array with both associative and numeric indices (the same as **OCI_ASSOC** + **OCI_NUM**). This is the default behavior.
- **OCI_ASSOC** - return an associative array (as oci_fetch_assoc() works).
- **OCI_NUM** - return a numeric array, (as oci_fetch_row() works).
- **OCI_RETURN_NULLS** - create empty elements for the **NULL** fields.
- **OCI_RETURN_LOBS** - return the value of a LOB of the descriptor.

Default *`mode`* is **OCI_BOTH**.

Return Values

Returns an array with both associative and numeric indices, or **FALSE** if there are no more rows in the *`statement`*.

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

Note

Oracle returns all field names in uppercase and associative indices in the result array will be uppercased too.

Examples

Example #16 - oci_fetch_array() with OCI_BOTH example

```
<?php
$connection = oci_connect("apelsin", "kanistra");

$query = "SELECT id, name FROM fruits";

$statement = oci_parse ($connection, $query);
oci_execute ($statement);

while ($row = oci_fetch_array ($statement, OCI_BOTH)) {
   echo $row[0]." and ".$row['ID']." is the same<br>";
   echo $row[1]." and ".$row['NAME']." is the same<br>";
}
?>
```

Example #17 - oci_fetch_array() with OCI_NUM example

```
<?php
$connection = oci_connect("user", "password");

$query = "SELECT id, name, lob_field FROM fruits";

$statement = oci_parse ($connection, $query);
oci_execute ($statement);

while ($row = oci_fetch_array ($statement, OCI_NUM)) {
   echo $row[0]."<br>";
   echo $row[1]."<br>";
   echo $row[2]->read(100)."<br>";  //this will output first 100 bytes from
LOB
}
?>
```

Example #18 - oci_fetch_array() with OCI_ASSOC example

```
<?php
```

```
$connection = oci_connect("user", "password");

$query = "SELECT id, name, lob_field FROM fruits";

$statement = oci_parse ($connection, $query);
oci_execute ($statement);

while ($row = oci_fetch_array ($statement, OCI_ASSOC)) {
   echo $row['ID']."<br>";
   echo $row['NAME']."<br>";
   echo $row['LOB_FIELD']."<br>";  //this will output "Object id #1"
}
?>
```

Example #19 - oci_fetch_array() with OCI_RETURN_LOBS example

```
<?php
$connection = oci_connect("user", "password");

$query = "SELECT id, name, lob_field FROM fruits";

$statement = oci_parse ($connection, $query);
oci_execute ($statement);

while ($row = oci_fetch_array ($statement, (OCI_NUM+OCI_RETURN_LOBS))) {
   echo $row[0]."<br>";
   echo $row[1]."<br>";
   echo $row['LOB_FIELD']."<br>";  //this will output LOB's content
}
?>
```

See Also

- oci_fetch_assoc()
- oci_fetch_object()
- oci_fetch_row()
- oci_fetch_all()

oci_fetch_assoc

oci_fetch_assoc -- Returns the next row from the result data as an associative array

Description

array **oci_fetch_assoc** (resource `$statement`)

Returns the next row from the result data as an associative array.

Calling oci_fetch_assoc() is identical to calling oci_fetch_array() with **OCI_ASSOC**.

A subsequent call to oci_fetch_assoc() will return the next row or **FALSE** if there are no more rows.

For details on the data type mapping performed by the oci8 driver, see the datatypes supported by the driver

Parameters

`statement`
A valid OCI statement identifier.

Return Values

Returns an associative array, or **FALSE** if there are no more rows in the *`statement`*.

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

Note
Oracle returns all field names in uppercase and associative indices in the result array will be uppercased too.

See Also

- oci_fetch_array()
- oci_fetch_object()

- oci_fetch_row()
- oci_fetch_all()

oci_fetch_object

oci_fetch_object -- Returns the next row from the result data as an object

Description

object **oci_fetch_object** (resource `$statement`)

Returns the next row from the result data as an object.

Subsequent calls to oci_fetch_object() will return the next row from the result or **FALSE** if there are no more rows.

For details on the data type mapping performed by the oci8 driver, see the datatypes supported by the driver

Parameters

`statement`
A valid OCI statement identifier.

Return Values

Returns an object, which attributes correspond to fields in statement, or **FALSE** if there are no more rows in the *`statement`*.

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

Note
Oracle returns all field names in uppercase and associative indices in the result object will be uppercased too.

See Also

- oci_fetch_array()
- oci_fetch_assoc()
- oci_fetch_row()

- oci_fetch_all()

oci_fetch_row

oci_fetch_row -- Returns the next row from the result data as a numeric array

Description

array **oci_fetch_row** (resource `$statement`)

Returns the next row from the result data as an indexed array.

Calling oci_fetch_row() is identical to calling oci_fetch_array() with **OCI_NUM**.

Subsequent calls to oci_fetch_row() will return the next row from the result data or **FALSE** if there are no more rows.

For details on the data type mapping performed by the oci8 driver, see the datatypes supported by the driver

Parameters

`statement`
A valid OCI statement identifier.

Return Values

Returns an indexed array with the field information, or **FALSE** if there are no more rows in the *`statement`*.

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

See Also

- oci_fetch_array()
- oci_fetch_assoc()
- oci_fetch_object()
- oci_fetch_all()

oci_fetch

oci_fetch -- Fetches the next row into result-buffer

Description

bool **oci_fetch** (resource `$statement`)

Fetches the next row (for SELECT statements) into the internal result-buffer.

For details on the data type mapping performed by the oci8 driver, see the datatypes supported by the driver

Parameters

`statement`
A valid OCI statement identifier.

Return Values

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

Notes

Note
In PHP versions before 5.0.0 you must use ocifetch() instead. This name still can be used, it was left as alias of oci_fetch() for downwards compatability. This, however, is deprecated and not recommended.

oci_field_is_null

oci_field_is_null -- Checks if the field is **NULL**

Description

bool **oci_field_is_null** (resource `$statement`, mixed `$field`)

Checks if the given *`field`* from the *`statement`* is **NULL**.

Parameters

`statement`
A valid OCI statement identifier.

`field`
Can be a field's index or a field's name (uppercased).

Return Values

Returns **TRUE** if *`field`* is **NULL**, **FALSE** otherwise.

Notes

Note
In PHP versions before 5.0.0 you must use ocicolumnisnull() instead. This name still can be used, it was left as alias of oci_field_is_null() for downwards compatability. This, however, is deprecated and not recommended.

oci_field_name

oci_field_name -- Returns the name of a field from the statement

Description

string **oci_field_name** (resource `$statement`, int `$field`)

Returns the name of the *`field`*.

Parameters

`statement`
A valid OCI statement identifier.

`field`
Can be the field's index (1-based) or name.

Return Values

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

Examples

Example #20 - oci_field_name() example

```
<?php
$conn = oci_connect("scott", "tiger");
$stmt = oci_parse($conn, "SELECT * FROM emp");
oci_execute($stmt);

echo "<table border=\"1\">";
echo "<tr>";
echo "<th>Name</th>";
echo "<th>Type</th>";
echo "<th>Length</th>";
echo "</tr>";

$ncols = oci_num_fields($stmt);

for ($i = 1; $i <= $ncols; $i++) {
   $column_name  = oci_field_name($stmt, $i);
   $column_type  = oci_field_type($stmt, $i);
   $column_size  = oci_field_size($stmt, $i);

   echo "<tr>";
   echo "<td>$column_name</td>";
   echo "<td>$column_type</td>";
```

```
   echo "<td>$column_size</td>";
   echo "</tr>";
}

echo "</table>\n";
oci_free_statement($stmt);
oci_close($conn);
?>
```

Notes

Note
In PHP versions before 5.0.0 you must use ocicolumnname() instead. This name still can be used, it was left as alias of oci_field_name() for downwards compatability. This, however, is deprecated and not recommended.

See Also

- oci_num_fields()
- oci_field_type()
- oci_field_size()

oci_field_precision

oci_field_precision -- Tell the precision of a field

Description

int **oci_field_precision** (resource `$statement`, int `$field`)

Returns precision of the *`field`*.

For FLOAT columns, precision is nonzero and scale is -127. If precision is 0, then column is NUMBER. Else it's NUMBER(precision, scale).

Parameters

`statement`
A valid OCI statement identifier.

`field`
Can be the field's index (1-based) or name.

Return Values

Returns the precision as an integer, or **FALSE** on errors.

Notes

Note
In PHP versions before 5.0.0 you must use ocicolumnprecision() instead. This name still can be used, it was left as alias of oci_field_precision() for downwards compatability. This, however, is deprecated and not recommended.

See Also

- oci_field_scale()
- oci_field_type()

oci_field_scale

oci_field_scale -- Tell the scale of the field

Description

int **oci_field_scale** (resource `$statement`, int `$field`)

Returns the scale of the column with *`field`* index.

For FLOAT columns, precision is nonzero and scale is -127. If precision is 0, then column is NUMBER. Else it's NUMBER(precision, scale).

Parameters

`statement`
A valid OCI statement identifier.

`field`
Can be the field's index (1-based) or name.

Return Values

Returns the scale as an integer, or **FALSE** on errors.

Notes

Note
In PHP versions before 5.0.0 you must use ocicolumnscale() instead. This name still can be used, it was left as alias of oci_field_scale() for downwards compatability. This, however, is deprecated and not recommended.

See Also

- oci_field_precision()
- oci_field_type()

oci_field_size

oci_field_size -- Returns field's size

Description

int **oci_field_size** (resource `$statement`, mixed `$field`)

Returns the size of a *`field`*.

Parameters

`statement`
A valid OCI statement identifier.

`field`
Can be the field's index (1-based) or name.

Return Values

Returns the size of a *`field`* in bytes, or **FALSE** on errors.

Examples

Example #21 - oci_field_size() example

```
<?php
$conn = oci_connect("scott", "tiger");
$stmt = oci_parse($conn, "SELECT * FROM emp");
oci_execute($stmt);

echo "<table border=\"1\">";
echo "<tr>";
echo "<th>Name</th>";
echo "<th>Type</th>";
echo "<th>Length</th>";
echo "</tr>";

$ncols = oci_num_fields($stmt);

for ($i = 1; $i <= $ncols; $i++) {
   $column_name  = oci_field_name($stmt, $i);
   $column_type  = oci_field_type($stmt, $i);
   $column_size  = oci_field_size($stmt, $i);
   echo "<tr>";
   echo "<td>$column_name</td>";
   echo "<td>$column_type</td>";
   echo "<td>$column_size</td>";
```

```
   echo "</tr>";
}

echo "</table>";

oci_free_statement($stmt);
oci_close($conn);
?>
```

Notes

Note
In PHP versions before 5.0.0 you must use ocicolumnsize() instead. This name still can be used, it was left as alias of oci_field_size() for downwards compatability. This, however, is deprecated and not recommended.

See Also

- oci_num_fields()
- oci_field_name()

oci_field_type_raw

oci_field_type_raw -- Tell the raw Oracle data type of the field

Description

int **oci_field_type_raw** (resource `$statement`, int `$field`)

Returns Oracle's raw data type of the *`field`*.

However, if you want to get field's type, then oci_field_type() will suit you better.

Parameters

`statement`
A valid OCI statement identifier.

`field`
Can be the field's index (1-based) or name.

Return Values

Returns Oracle's raw data type as a string, or **FALSE** on errors.

Notes

Note
In PHP versions before 5.0.0 you must use ocicolumntyperaw() instead. This name still can be used, it was left as alias of oci_field_type_raw() for downwards compatability. This, however, is deprecated and not recommended.

oci_field_type

oci_field_type -- Returns field's data type

Description

mixed **oci_field_type** (resource `$statement`, int `$field`)

Returns a field's data type.

Parameters

`statement`
A valid OCI statement identifier.

`field`
Can be the field's index (1-based) or name.

Return Values

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

Return Values

Example #22 - oci_field_type() example

```
<?php
$conn = oci_connect("scott", "tiger");
$stmt = oci_parse($conn, "SELECT * FROM emp");
oci_execute($stmt);

echo "<table border=\"1\">";
echo "<tr>";
echo "<th>Name</th>";
echo "<th>Type</th>";
echo "<th>Length</th>";
echo "</tr>";

$ncols = oci_num_fields($stmt);

for ($i = 1; $i <= $ncols; $i++) {
   $column_name  = oci_field_name($stmt, $i);
   $column_type  = oci_field_type($stmt, $i);
   $column_size  = oci_field_size($stmt, $i);

   echo "<tr>";
   echo "<td>$column_name</td>";
   echo "<td>$column_type</td>";
```

```
    echo "<td>$column_size</td>";
    echo "</tr>";
}

echo "</table>\n";

oci_free_statement($stmt);
oci_close($conn);
?>
```

Notes

Note
In PHP versions before 5.0.0 you must use ocicolumntype() instead. This name still can be used, it was left as alias of oci_field_type() for downwards compatability. This, however, is deprecated and not recommended.

See Also

- oci_num_fields()
- oci_field_name()
- oci_field_size()

oci_free_statement

oci_free_statement -- Frees all resources associated with statement or cursor

Description

bool **oci_free_statement** (resource `$statement`)

Frees resources associated with Oracle's cursor or statement, which was received from as a result of oci_parse() or obtained from Oracle.

Parameters

`statement`
A valid OCI statement identifier.

Return Values

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

oci_internal_debug

oci_internal_debug -- Enables or disables internal debug output

Description

void **oci_internal_debug** (bool `$onoff`)

Enables or disables internal debug output.

Parameters

`onoff`
Set this to **FALSE** to turn debug output off or **TRUE** to turn it on.

Return Values

No value is returned.

Notes

Note
In PHP versions before 5.0.0 you must use ociinternaldebug() instead. This name still can be used, it was left as alias of oci_internal_debug() for downwards compatability. This, however, is deprecated and not recommended.

OCI-Lob->append

OCI-Lob->append -- Appends data from the large object to another large object

Description

OCI-Lob

bool **append** (OCI-Lob `$lob_from`)

Appends data from the large object to the end of another large object.

Writing to the large object with this method will fail if buffering was previously enabled. You must disable buffering before appending. You may need to flush buffers with OCI-Lob->flush before disabling buffering.

Parameters

`lob_from`
The copied LOB.

Return Values

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

See Also

- OCI-Lob->flush
- OCI-Lob->setBuffering
- OCI-Lob->getBuffering

OCI-Lob->close

OCI-Lob->close -- Closes LOB descriptor

Description

OCI-Lob

bool **close** (void)

Closes descriptor of LOB or FILE. This function should be used only with OCI-Lob->writeTemporary.

Return Values

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

See Also

- OCI-Lob->writeTemporary

oci_lob_copy

oci_lob_copy -- Copies large object

Description

bool **oci_lob_copy** (OCI-Lob `$lob_to`, OCI-Lob `$lob_from` [, int `$length`])

Copies a large object or a part of a large object to another large object. Old LOB-recipient data will be overwritten.

If you need to copy a particular part of a LOB to a particular position of a LOB, use oci_lob_seek() to move LOB internal pointers.

Parameters

`lob_to`
: The destination LOB.

`lob_from`
: The copied LOB.

`length`
: Indicates the length of data to be copied.

Return Values

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

OCI-Lob->eof

OCI-Lob->eof -- Tests for end-of-file on a large object's descriptor

Description

OCI-Lob

bool **eof** (void)

Tells whether the internal pointer of large object is at the end of LOB.

Return Values

Returns **TRUE** if internal pointer of large object is at the end of LOB. Otherwise returns **FALSE**.

See Also

- OCI-Lob->size

OCI-Lob->erase

OCI-Lob->erase -- Erases a specified portion of the internal LOB data

Description

OCI-Lob

int **erase** ([int `$offset` [, int `$length`]])

Erases a specified portion of the internal LOB data starting at a specified *`offset`*. If called without parameters, it erases all LOB data.

For BLOBs, erasing means that the existing LOB value is overwritten with zero-bytes. For CLOBs, the existing LOB value is overwritten with spaces.

Parameters

`offset`

`length`

Return Values

Returns the actual number of characters/bytes erased or **FALSE** in case of error.

See Also

- OCI-Lob->truncate

OCI-Lob->export

OCI-Lob->export -- Exports LOB's contents to a file

Description

OCI-Lob

bool **export** (string `$filename` [, int `$start` [, int `$length`]])

Exports LOB contents to a file.

Parameters

`filename`
: Path to the file.

`start`
: Indicates from where to start exporting.

`length`
: Indicates the length of data to be exported.

Return Values

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

See Also

- OCI-Lob->import

OCI-Lob->flush

OCI-Lob->flush -- Flushes/writes buffer of the LOB to the server

Description

OCI-Lob

bool **flush** ([int `$flag`])

OCI-Lob->flush() actually writes data to the server.

Parameters

`flag`
: By default, resources are not freed, but using flag **OCI_LOB_BUFFER_FREE** you can do it explicitly. Be sure you know what you're doing - next read/write operation to the same part of LOB will involve a round-trip to the server and initialize new buffer resources. It is recommended to use **OCI_LOB_BUFFER_FREE** flag only when you are not going to work with the LOB anymore.

Return Values

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

Returns **FALSE** if buffering was not enabled or an error occurred.

See Also

- OCI-Lob->getBuffering
- OCI-Lob->setBuffering

OCI-Lob->free

OCI-Lob->free -- Frees resources associated with the LOB descriptor

Description

OCI-Lob

bool **free** (void)

Frees resources associated with the descriptor, previously allocated with oci_new_descriptor().

Return Values

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

OCI-Lob->getBuffering

OCI-Lob->getBuffering -- Returns current state of buffering for the large object

Description

OCI-Lob

bool **getBuffering** (void)

Tells whether the buffering for the large object is on or off.

Return Values

Returns **FALSE** if buffering for the large object is off and **TRUE** if buffering is used.

See Also

- OCI-Lob->setBuffering

OCI-Lob->import

OCI-Lob->import -- Imports file data to the LOB

Description

OCI-Lob

bool **import** (string `$filename`)

Writes data from the *`filename`* in to the current position of large object.

Parameters

`filename`
: Path to the file.

Return Values

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

See Also

- OCI-Lob->export
- OCI-Lob->write

oci_lob_is_equal

oci_lob_is_equal -- Compares two LOB/FILE locators for equality

Description

bool **oci_lob_is_equal** (OCI-Lob `$lob1`, OCI-Lob `$lob2`)

Compares two LOB/FILE locators.

Parameters

`lob1`
A LOB identifier.

`lob2`
A LOB identifier.

Return Values

Returns **TRUE** if these objects are equal, **FALSE** otherwise.

OCI-Lob->load

OCI-Lob->load -- Returns large object's contents

Description

OCI-Lob

string **load** (void)

Returns large object's contents. As script execution is terminated when the memory_limit is reached, ensure that the LOB does not exceed this limit. In most cases it's recommended to use OCI-Lob->read instead.

Return Values

Returns the contents of the object, or **FALSE** on errors.

See Also

- OCI-Lob->read

OCI-Lob->read

OCI-Lob->read -- Reads part of the large object

Description

OCI-Lob

string **read** (int `$length`)

Reads *`length`* bytes from the current position of LOB's internal pointer.

Reading stops when *`length`* bytes have been read or end of the large object is reached. Internal pointer of the large object will be shifted on the amount of bytes read.

Parameters

`length`
: The length of data to read, in bytes.

Return Values

Returns the contents as a string, or **FALSE** in case of error.

See Also

- OCI-Lob->load
- OCI-Lob->write

OCI-Lob->rewind

OCI-Lob->rewind -- Moves the internal pointer to the beginning of the large object

Description

OCI-Lob

bool **rewind** (void)

Sets the internal pointer to the beginning of the large object.

Return Values

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

See Also

- OCI-Lob->seek
- OCI-Lob->tell

OCI-Lob->save

OCI-Lob->save -- Saves data to the large object

Description

OCI-Lob

bool **save** (string `$data` [, int `$offset`])

Saves *`data`* to the large object.

Parameters

`data`
: The data to be saved.

`offset`
: Can be used to indicate offset from the beginning of the large object.

Return Values

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

See Also

- OCI-Lob->write
- OCI-Lob->import

OCI-Lob->saveFile

OCI-Lob->saveFile -- Alias of oci_lob_import()

Description

This function is an alias of: oci_lob_import().

OCI-Lob->seek

OCI-Lob->seek -- Sets the internal pointer of the large object

Description

OCI-Lob

bool **seek** (int `$offset` [, int `$whence`])

Sets the internal pointer of the large object.

Parameters

`offset`
: Indicates the amount of bytes, on which internal pointer should be moved from the position, pointed by *`whence`*.

`whence`
: May be one of:

- **OCI_SEEK_SET** - sets the position equal to *`offset`*
- **OCI_SEEK_CUR** - adds *`offset`* bytes to the current position
- **OCI_SEEK_END** - adds *`offset`* bytes to the end of large object (use negative value to move to a position before the end of large object)

Return Values

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

See Also

- OCI-Lob->rewind
- OCI-Lob->tell
- OCI-Lob->eof

OCI-Lob->setBuffering

OCI-Lob->setBuffering -- Changes current state of buffering for the large object

Description

OCI-Lob

bool **setBuffering** (bool `$on_off`)

Sets the buffering for the large object, depending on the value of the *`on_off`* parameter.

Use of this function may provide performance improvements by buffering small reads and writes of LOBs by reducing the number of network round-trips and LOB versions. oci_lob_flush() should be used to flush buffers, when you have finished working with the large object.

Parameters

`on_off`
: **TRUE** for on and **FALSE** for off.

Return Values

Returns **TRUE** on success or **FALSE** on failure. Repeated calls to this method with the same flag will return **TRUE**.

See Also

- OCI-Lob->getBuffering

OCI-Lob->size

OCI-Lob->size -- Returns size of large object

Description

OCI-Lob

int **size** (void)

Gets the size of the large object.

Return Values

Returns length of large object value or **FALSE** in case of error. Empty objects have zero length.

OCI-Lob->tell

OCI-Lob->tell -- Returns current position of internal pointer of large object

Description

OCI-Lob

int **tell** (void)

Gets the current position of a LOB's internal pointer.

Return Values

Returns current position of a LOB's internal pointer or **FALSE** if an error occurred.

See Also

- OCI-Lob->rewind
- OCI-Lob->size
- OCI-Lob->eof

OCI-Lob->truncate

OCI-Lob->truncate -- Truncates large object

Description

OCI-Lob

bool **truncate** ([int `$length`])

Truncates the LOB.

Parameters

`length`
: If provided, this method will truncate the LOB to *`length`* bytes. Otherwise, it will completrely purge the LOB.

Return Values

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

See Also

- OCI-Lob->erase

OCI-Lob->write

OCI-Lob->write -- Writes data to the large object

Description

OCI-Lob

int **write** (string `$data` [, int `$length`])

Writes data from the parameter *`data`* into the current position of LOB's internal pointer.

Parameters

`data`
: The data to write in the LOB.

`length`
: If this parameter is given, writing will stop after *`length`* bytes have been written or the end of *`data`* is reached, whichever comes first.

Return Values

Returns the number of bytes written or **FALSE** in case of error.

See Also

- OCI-Lob->read

OCI-Lob->writeTemporary

OCI-Lob->writeTemporary -- Writes temporary large object

Description

OCI-Lob

bool **writeTemporary** (string `$data` [, int `$lob_type`])

Creates a temporary large object and writes *`data`* to it.

You should use OCI-Lob->close when you are done with this object.

Parameters

`data`
: The data to write.

`lob_type`
: Can be one of the following:

- **OCI_TEMP_BLOB** is used to create temporary BLOBs
- **OCI_TEMP_CLOB** (default value) is used to create temporary CLOBs

Return Values

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

See Also

- OCI-Lob->close

OCI-Lob->writeToFile

OCI-Lob->writeToFile -- Alias of oci_lob_export()

Description

This function is an alias of: oci_lob_export().

oci_new_collection

oci_new_collection -- Allocates new collection object

Description

OCI-Collection **oci_new_collection** (resource `$connection`, string `$tdo` [, string `$schema`])

Allocates a new collection object.

Parameters

`connection`
An Oracle connection identifier, returned by oci_connect() or oci_pconnect().

`tdo`
Should be a valid named type (uppercase).

`schema`
Should point to the scheme, where the named type was created. The name of the current user is the default value.

Return Values

Returns a new OCICollection object or **FALSE** on error.

Notes

Note
In PHP versions before 5.0.0 you must use ocinewcollection() instead. This name still can be used, it was left as alias of oci_new_collection() for downwards compatability. This, however, is deprecated and not recommended.

oci_new_connect

oci_new_connect -- Establishes a new connection to the Oracle server

Description

resource **oci_new_connect** (string `$username`, string `$password` [, string `$db` [, string `$charset` [, int `$session_mode`]]])

Establishes a new connection to an Oracle server and logs on.

Unlike oci_connect() and oci_pconnect(), oci_new_connect() does not cache connections and will always return a brand-new freshly opened connection handle. This is useful if your application needs transactional isolation between two sets of queries.

Parameters

`username`
The Oracle user name.

`password`
The password for *`username`*.

`db`
This optional parameter can either contain the name of the local Oracle instance or the name of the entry in *tnsnames.ora*. If the not specified, PHP uses environment variables *ORACLE_SID* and *TWO_TASK* to determine the name of local Oracle instance and location of *tnsnames.ora* accordingly.

`charset`
Using Oracle server version 9.2 and greater, you can indicate *`charset`* by parameter, which will be used in the new connection. If you're using Oracle server < 9.2, this parameter will be ignored and the *NLS_LANG* environment variable will be used instead.

`session_mode`
This parameter is available since version 1.1 and accepts the following values: **OCI_DEFAULT**, **OCI_SYSOPER** and **OCI_SYSDBA**. If either **OCI_SYSOPER** or **OCI_SYSDBA** were specified, this function will try to establish privileged connection using external credentials. Privileged connections are disabled by default. To enable them you need to set oci8.privileged_connect to *On*.

Return Values

Returns a connection identifier or **FALSE** on error.

Examples

The following demonstrates how you can separate connections.

Example #23 - oci_new_connect() example

```
<?php
echo "<html><pre>";
$db = "";

$c1 = oci_connect("scott", "tiger", $db);
$c2 = oci_new_connect("scott", "tiger", $db);

function create_table($conn)
{
 $stmt = oci_parse($conn, "create table scott.hallo (test
varchar2(64))");
 oci_execute($stmt);
 echo $conn . " created table\n\n";
}

function drop_table($conn)
{
 $stmt = oci_parse($conn, "drop table scott.hallo");
 oci_execute($stmt);
 echo $conn . " dropped table\n\n";
}

function insert_data($conn)
{
 $stmt = oci_parse($conn, "insert into scott.hallo
           values('$conn' || ' ' || to_char(sysdate,'DD-MON-YY
HH24:MI:SS'))");
 oci_execute($stmt, OCI_DEFAULT);
 echo $conn . " inserted hallo\n\n";
}

function delete_data($conn)
{
 $stmt = oci_parse($conn, "delete from scott.hallo");
 oci_execute($stmt, OCI_DEFAULT);
 echo $conn . " deleted hallo\n\n";
}

function commit($conn)
{
 oci_commit($conn);
 echo $conn . " committed\n\n";
}

function rollback($conn)
{
 oci_rollback($conn);
 echo $conn . " rollback\n\n";
}

function select_data($conn)
{
 $stmt = oci_parse($conn, "select * from scott.hallo");
 oci_execute($stmt, OCI_DEFAULT);
 echo $conn . "----selecting\n\n";
 while (oci_fetch($stmt)) {
```

```
   echo $conn . " <" . oci_result($stmt, "TEST") . ">\n\n";
 }
 echo $conn . "----done\n\n";
}

create_table($c1);
insert_data($c1);

select_data($c1);
select_data($c2);

rollback($c1);

select_data($c1);
select_data($c2);

insert_data($c2);
commit($c2);

select_data($c1);

delete_data($c1);
select_data($c1);
select_data($c2);
commit($c1);

select_data($c1);
select_data($c2);

drop_table($c1);
echo "</pre></html>";
?>
```

Notes

Note

If you're using PHP with Oracle Instant Client, you can use easy connect naming method described here: » http://download-west.oracle.com/docs/cd/B12037_01/network.101/b10775/naming.htm#i498306. Basically this means you can specify "//db_host[:port]/database_name" as database name. But if you want to use the old way of naming you *must* set either **ORACLE_HOME** or **TNS_ADMIN**.

Note

In PHP versions before 5.0.0 you must use ocinlogon() instead. This name still can be used, it was left as alias of oci_new_connect() for downwards compatability. This, however, is deprecated and not recommended.

See Also

- oci_connect()
- oci_pconnect()

oci_new_cursor

oci_new_cursor -- Allocates and returns a new cursor (statement handle)

Description

resource **oci_new_cursor** (resource `$connection`)

Allocates a new statement handle on the specified connection.

Parameters

`connection`
An Oracle connection identifier, returned by oci_connect() or oci_pconnect().

Return Values

Returns a new statement handle, or **FALSE** on error.

Examples

Example #24 - Using REF CURSOR in an Oracle's stored procedure

```
<?php
// suppose your stored procedure info.output returns a ref cursor in :data

$conn = oci_connect("scott", "tiger");
$curs = oci_new_cursor($conn);
$stmt = oci_parse($conn, "begin info.output(:data); end;");

oci_bind_by_name($stmt, "data", $curs, -1, OCI_B_CURSOR);
oci_execute($stmt);
oci_execute($curs);

while ($data = oci_fetch_row($curs)) {
   var_dump($data);
}

oci_free_statement($stmt);
oci_free_statement($curs);
oci_close($conn);
?>
```

Example #25 - Using REF CURSOR in an Oracle's select statement

```
<?php
echo "<html><body>";
$conn = oci_connect("scott", "tiger");
$count_cursor = "CURSOR(select count(empno) num_emps from emp " .
                "where emp.deptno = dept.deptno) as EMPCNT from dept";
$stmt = oci_parse($conn, "select deptno,dname,$count_cursor");

oci_execute($stmt);
echo "<table border=\"1\">";
echo "<tr>";
echo "<th>DEPT NAME</th>";
echo "<th>DEPT #</th>";
echo "<th># EMPLOYEES</th>";
echo "</tr>";

while ($data = oci_fetch_assoc($stmt)) {
   echo "<tr>";
   $dname  = $data["DNAME"];
   $deptno = $data["DEPTNO"];
   echo "<td>$dname</td>";
   echo "<td>$deptno</td>";
   oci_execute($data["EMPCNT"]);
   while ($subdata = oci_fetch_assoc($data["EMPCNT"])) {
       $num_emps = $subdata["NUM_EMPS"];
       echo  "<td>$num_emps</td>";
   }
   echo "</tr>";
}
echo "</table>";
echo "</body></html>";
oci_free_statement($stmt);
oci_close($conn);
?>
```

Notes

Note

In PHP versions before 5.0.0 you must use ocinewcursor() instead. This name still can be used, it was left as alias of oci_new_cursor() for downwards compatability. This, however, is deprecated and not recommended.

oci_new_descriptor

oci_new_descriptor -- Initializes a new empty LOB or FILE descriptor

Description

OCI-Lob **oci_new_descriptor** (resource `$connection` [, int `$type`])

Allocates resources to hold descriptor or LOB locator.

Parameters

`connection`
An Oracle connection identifier, returned by oci_connect() or oci_pconnect().

`type`
Valid values for *`type`* are: **OCI_D_FILE**, **OCI_D_LOB** and **OCI_D_ROWID**.

Return Values

Returns a new LOB or FILE descriptor on success, **FALSE** on error.

Examples

Example #26 - oci_new_descriptor() example

```
<?php
/* This script is designed to be called from a HTML form.
* It expects $user, $password, $table, $where, and $commitsize
* to be passed in from the form.  The script then deletes
* the selected rows using the ROWID and commits after each
* set of $commitsize rows. (Use with care, there is no rollback)
*/
$conn = oci_connect($user, $password);
$stmt = oci_parse($conn, "select rowid from $table $where");
$rowid = oci_new_descriptor($conn, OCI_D_ROWID);
oci_define_by_name($stmt, "ROWID", $rowid);
oci_execute($stmt);
while (oci_fetch($stmt)) {
   $nrows = oci_num_rows($stmt);
   $delete = oci_parse($conn, "delete from $table where ROWID = :rid");
   oci_bind_by_name($delete, ":rid", $rowid, -1, OCI_B_ROWID);
   oci_execute($delete);
   echo "$nrows\n";
   if (($nrows % $commitsize) == 0) {
       oci_commit($conn);
   }
}
```

```
$nrows = oci_num_rows($stmt);
echo "$nrows deleted...\n";
oci_free_statement($stmt);
oci_close($conn);
?>

<?php
   /* This script demonstrates file upload to LOB columns
    * The formfield used for this example looks like this
    * <form action="upload.php" method="post" enctype="multipart/form-data">
    * <input type="file" name="lob_upload" />
    * ...
    */
 if (!isset($lob_upload) || $lob_upload == 'none'){
?>
<form action="upload.php" method="post" enctype="multipart/form-data">
Upload file: <input type="file" name="lob_upload" /><br />
<input type="submit" value="Upload" /> - <input type="reset" value="Reset"
/>
</form>
<?php
 } else {

    // $lob_upload contains the temporary filename of the uploaded file

    // see also the features section on file upload,
    // if you would like to use secure uploads

    $conn = oci_connect($user, $password);
    $lob = oci_new_descriptor($conn, OCI_D_LOB);
    $stmt = oci_parse($conn, "insert into $table (id, the_blob)
              values(my_seq.NEXTVAL, EMPTY_BLOB()) returning the_blob into
:the_blob");
    oci_bind_by_name($stmt, ':the_blob', $lob, -1, OCI_B_BLOB);
    oci_execute($stmt, OCI_DEFAULT);
    if ($lob->savefile($lob_upload)){
       oci_commit($conn);
       echo "Blob successfully uploaded\n";
    }else{
       echo "Couldn't upload Blob\n";
    }
    oci_free_descriptor($lob);
    oci_free_statement($stmt);
    oci_close($conn);
 }
?>
```

Example #27 - oci_new_descriptor() **example**

```
<?php
/* Calling PL/SQL stored procedures which contain clobs as input
* parameters (PHP 4 >= 4.0.6).
* Example PL/SQL stored procedure signature is:
*
* PROCEDURE save_data
*   Argument Name                  Type                      In/Out Default?
*   ------------------------------ ------------------------- ------ --------
*   KEY                            NUMBER(38)                IN
```

```
*   DATA                            CLOB                    IN
*
*/

$conn = oci_connect($user, $password);
$stmt = oci_parse($conn, "begin save_data(:key, :data); end;");
$clob = oci_new_descriptor($conn, OCI_D_LOB);
oci_bind_by_name($stmt, ':key', $key);
oci_bind_by_name($stmt, ':data', $clob, -1, OCI_B_CLOB);
$clob->write($data);
oci_execute($stmt, OCI_DEFAULT);
oci_commit($conn);
$clob->free();
oci_free_statement($stmt);
?>
```

Notes

Note
In PHP versions before 5.0.0 you must use ocinewdescriptor() instead. This name still can be used, it was left as alias of oci_new_descriptor() for downwards compatability. This, however, is deprecated and not recommended.

oci_num_fields

oci_num_fields -- Returns the number of result columns in a statement

Description

int **oci_num_fields** (resource `$statement`)

Gets the number of columns in the given *`statement`*.

Parameters

`statement`
A valid OCI statement identifier.

Return Values

Returns the number of columns as an integer, or **FALSE** on errors.

Examples

Example #28 - oci_num_fields() **example**

```
<?php
$conn = oci_connect("scott", "tiger");
$stmt = oci_parse($conn, "select * from emp");

oci_execute($stmt);

while (oci_fetch($stmt)) {
   echo "\n";
   $ncols = oci_num_fields($stmt);
   for ($i = 1; $i <= $ncols; $i++) {
       $column_name  = oci_field_name($stmt, $i);
       $column_value = oci_result($stmt, $i);
       echo $column_name . ': ' . $column_value . "\n";
   }
   echo "\n";
}

oci_free_statement($stmt);
oci_close($conn);
?>
```

Notes

Note
In PHP versions before 5.0.0 you must use ocinumcols() instead. This name still can be used, it was left as alias of oci_num_fields() for downwards compatability. This, however, is deprecated and not recommended.

oci_num_rows

oci_num_rows -- Returns number of rows affected during statement execution

Description

int **oci_num_rows** (resource `$statement`)

Gets the number of rows affected during statement execution.

Parameters

`statement`
A valid OCI statement identifier.

Return Values

Returns the number of rows affected as an integer, or **FALSE** on errors.

Examples

Example #29 - oci_num_rows() **example**

```
<?php
$conn = oci_connect("scott", "tiger");

$stmt = oci_parse($conn, "create table emp2 as select * from emp");
oci_execute($stmt);
echo oci_num_rows($stmt) . " rows inserted.<br />";
oci_free_statement($stmt);

$stmt = oci_parse($conn, "delete from emp2");
oci_execute($stmt, OCI_DEFAULT);
echo oci_num_rows($stmt) . " rows deleted.<br />";
oci_commit($conn);
oci_free_statement($stmt);

$stmt = oci_parse($conn, "drop table emp2");
oci_execute($stmt);
oci_free_statement($stmt);

oci_close($conn);
?>
```

Notes

Note

This function *does not* return number of rows selected! For SELECT statements this function will return the number of rows, that were fetched to the buffer with **oci_fetch*()** functions.

Note

In PHP versions before 5.0.0 you must use ocirowcount() instead. This name still can be used, it was left as alias of oci_num_rows() for downwards compatability. This, however, is deprecated and not recommended.

oci_parse

oci_parse -- Prepares Oracle statement for execution

Description

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

Prepares the *`query`* using *`connection`* and returns the statement identifier, which can be used with oci_bind_by_name(), oci_execute() and other functions.

Parameters

`connection`
An Oracle connection identifier, returned by oci_connect() or oci_pconnect().

`query`
The SQL query.

Return Values

Returns a statement handler on success, or **FALSE** on error.

Notes

Note
This function *does not* validate *`query`*. The only way to find out if *`query`* is valid SQL or PL/SQL statement - is to execute it.

Note
In PHP versions before 5.0.0 you must use ociparse() instead. This name still can be used, it was left as alias of oci_parse() for downwards compatability. This, however, is deprecated and not recommended.

oci_password_change

oci_password_change -- Changes password of Oracle's user

Description

bool **oci_password_change** (resource `$connection`, string `$username`, string `$old_password`, string `$new_password`)

resource **oci_password_change** (string `$dbname`, string `$username`, string `$old_password`, string `$new_password`)

Changes password for user with *`username`*.

Parameters

`connection`
An Oracle connection identifier, returned by oci_connect() or oci_pconnect().

`username`
The Oracle user name.

`old_password`
The old password.

`new_password`
The new password to be set.

`dbname`
The database name.

Return Values

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

Notes

Note
The second oci_password_change() syntax is available since version 1.1.

Note
In PHP versions before 5.0.0 you must use **ocipasswordchange()** instead. This name

still can be used, it was left as alias of oci_password_change() for downwards compatability. This, however, is deprecated and not recommended.

oci_pconnect

oci_pconnect -- Connect to an Oracle database using a persistent connection

Description

resource **oci_pconnect** (string `$username`, string `$password` [, string `$db` [, string `$charset` [, int `$session_mode`]]])

Creates a persistent connection to an Oracle server and logs on.

Persistent connections are cached and re-used between requests, resulting in reduced overhead on each page load; a typical PHP application will have a single persistent connection open against an Oracle server per Apache child process (or PHP FastCGI/CGI process). See the Persistent Database Connections section for more information.

Parameters

`username`
The Oracle user name.

`password`
The password for *`username`*.

`db`
This optional parameter can either contain the name of the local Oracle instance or the name of the entry in *tnsnames.ora*. If the not specified, PHP uses environment variables *ORACLE_SID* and *TWO_TASK* to determine the name of local Oracle instance and location of *tnsnames.ora* accordingly.

`charset`
Using Oracle server version 9.2 and greater, you can indicate *`charset`* by parameter, which will be used in the new connection. If you're using Oracle server < 9.2, this parameter will be ignored and the *NLS_LANG* environment variable will be used instead.

`session_mode`
This parameter is available since version 1.1 and accepts the following values: **OCI_DEFAULT**, **OCI_SYSOPER** and **OCI_SYSDBA**. If either **OCI_SYSOPER** or **OCI_SYSDBA** were specified, this function will try to establish privileged connection using external credentials. Privileged connections are disabled by default. To enable them you need to set oci8.privileged_connect to *On*.

Return Values

Returns a connection identifier or **FALSE** on error.

Notes

Note
Starting with version 1.1 of the oci8 extension, the lifetime and maximum amount of persistent Oracle connections can be tuned by setting the following configuration values: oci8.persistent_timeout, oci8.ping_interval and oci8.max_persistent.

Note
If you're using PHP with Oracle Instant Client, you can use easy connect naming method described here: » http://download-west.oracle.com/docs/cd/B12037_01/network.101/b10775/naming.htm#i498306. Basically this means you can specify "//db_host[:port]/database_name" as database name. But if you want to use the old way of naming you *must* set either **ORACLE_HOME** or **TNS_ADMIN**.

Note
In PHP versions before 5.0.0 you must use ociplogon() instead. This name still can be used, it was left as alias of oci_pconnect() for downwards compatability. This, however, is deprecated and not recommended.

See Also

- oci_connect()
- oci_new_connect()

oci_result

oci_result -- Returns field's value from the fetched row

Description

mixed **oci_result** (resource `$statement`, mixed `$field`)

Returns the data from *`field`* in the current row, fetched by oci_fetch().

For details on the data type mapping performed by the oci8 driver, see the datatypes supported by the driver

Parameters

`statement`

`field`
Can be either use the column number (1-based) or the column name (in uppercase).

Return Values

Returns everything as strings except for abstract types (ROWIDs, LOBs and FILEs). Returns **FALSE** on error.

Notes

Note
In PHP versions before 5.0.0 you must use ociresult() instead. This name still can be used, it was left as alias of oci_result() for downwards compatability. This, however, is deprecated and not recommended.

See Also

- oci_fetch_array()
- oci_fetch_assoc()
- oci_fetch_object()
- oci_fetch_row()
- oci_fetch_all()

oci_rollback

oci_rollback -- Rolls back outstanding transaction

Description

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

Rolls back all outstanding statements for the Oracle *`connection`*.

Parameters

`connection`
An Oracle connection identifier, returned by oci_connect() or oci_pconnect().

Return Values

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

Notes

Note
Transactions are automatically rolled back when you close the connection, or when the script ends, whichever is soonest. You need to explicitly call oci_commit() to commit the transaction, or oci_rollback() to abort it.

Note
In PHP versions before 5.0.0 you must use ocirollback() instead. This name still can be used, it was left as alias of oci_rollback() for downwards compatability. This, however, is deprecated and not recommended.

See Also

- oci_commit()

oci_server_version

oci_server_version -- Returns server version

Description

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

Returns a string with version information of the Oracle server, which uses the provided *`connection`*.

Parameters

`connection`

Return Values

Returns the version information as a string or **FALSE** on error.

Examples

Example #30 - oci_server_version() example

```
<?php
   $conn = oci_connect("scott", "tiger");
   echo "Server Version: " . oci_server_version($conn);
   oci_close($conn);
?>
```

Notes

Note

In PHP versions before 5.0.0 you must use ociserverversion() instead. This name still can be used, it was left as alias of oci_server_version() for downwards compatability. This, however, is deprecated and not recommended.

oci_set_prefetch

oci_set_prefetch -- Sets number of rows to be prefetched

Description

bool **oci_set_prefetch** (resource `$statement`, int `$rows`)

Sets the number of rows to be prefetched after successful call to oci_execute().

Parameters

`statement`

`rows`

Return Values

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

Notes

Note
In PHP versions before 5.0.0 you must use ocisetprefetch() instead. This name still can be used, it was left as alias of oci_set_prefetch() for downwards compatability. This, however, is deprecated and not recommended.

See Also

- oci8_.default_prefetch ini option

oci_statement_type

oci_statement_type -- Returns the type of an OCI statement

Description

string **oci_statement_type** (resource `$statement`)

Returns the type of the provided OCI *`statement`*.

Parameters

`statement`
A valid OCI statement identifier.

Return Values

Returns the query type of *`statement`* as one of the following values:

- *SELECT*
- *UPDATE*
- *DELETE*
- *INSERT*
- *CREATE*
- *DROP*
- *ALTER*
- *BEGIN*
- *DECLARE*
- *CALL* (since PHP 5.2.1 and OCI8 1.2.3)
- *UNKNOWN*

Returns **FALSE** on error.

Examples

Example #31 - oci_statement_type() example

```
<?php
   $conn = oci_connect("scott", "tiger");
   $sql  = "delete from emp where deptno = 10";
```

```
   $stmt = oci_parse($conn, $sql);
   if (oci_statement_type($stmt) == "DELETE") {
       die("You are not allowed to delete from this table<br />");
   }

   oci_close($conn);
?>
```

Notes

Note
In PHP versions before 5.0.0 you must use ocistatementtype() instead. This name still can be used, it was left as alias of oci_statement_type() for downwards compatability. This, however, is deprecated and not recommended.

ocibindbyname

ocibindbyname -- Alias of oci_bind_by_name()

Description

This function is an alias of: oci_bind_by_name().

ocicancel

ocicancel -- Alias of oci_cancel()

Description

This function is an alias of: oci_cancel().

ocicloselob

ocicloselob -- Alias of OCI-Lob->close

Description

This function is an alias of: OCI-Lob->close.

ocicollappend

ocicollappend -- Alias of OCI-Collection->append

Description

This function is an alias of: OCI-Collection->append.

ocicollassign

ocicollassign -- Alias of OCI-Collection->assign

Description

This function is an alias of: OCI-Collection->assign.

ocicollassignelem

ocicollassignelem -- Alias of OCI-Collection->assignElem

Description

This function is an alias of: OCI-Collection->assignElem.

ocicollgetelem

ocicollgetelem -- Alias of OCI-Collection->getElem

Description

This function is an alias of: OCI-Collection->getElem.

ocicollmax

ocicollmax -- Alias of OCI-Collection->max

Description

This function is an alias of: OCI-Collection->max.

ocicollsize

ocicollsize -- Alias of OCI-Collection->size

Description

This function is an alias of: OCI-Collection->size.

ocicolltrim

ocicolltrim -- Alias of OCI-Collection->trim

Description

This function is an alias of: OCI-Collection->trim.

ocicolumnisnull

ocicolumnisnull -- Alias of oci_field_is_null()

Description

This function is an alias of: oci_field_is_null().

ocicolumnname

ocicolumnname -- Alias of oci_field_name()

Description

This function is an alias of: oci_field_name().

ocicolumnprecision

ocicolumnprecision -- Alias of oci_field_precision()

Description

This function is an alias of: oci_field_precision().

ocicolumnscale

ocicolumnscale -- Alias of oci_field_scale()

Description

This function is an alias of: oci_field_scale().

ocicolumnsize

ocicolumnsize -- Alias of oci_field_size()

Description

This function is an alias of: oci_field_size().

ocicolumntype

ocicolumntype -- Alias of oci_field_type()

Description

This function is an alias of: oci_field_type().

ocicolumntyperaw

ocicolumntyperaw -- Alias of oci_field_type_raw()

Description

This function is an alias of: oci_field_type_raw().

ocicommit

ocicommit -- Alias of oci_commit()

Description

This function is an alias of: oci_commit().

ocidefinebyname

ocidefinebyname -- Alias of oci_define_by_name()

Description

This function is an alias of: oci_define_by_name().

ocierror

ocierror -- Alias of oci_error()

Description

This function is an alias of: oci_error().

ociexecute

ociexecute -- Alias of oci_execute()

Description

This function is an alias of: oci_execute().

ocifetch

ocifetch -- Alias of oci_fetch()

Description

This function is an alias of: oci_fetch().

ocifetchinto

ocifetchinto -- Fetches the next row into an array (deprecated)

Description

int **ocifetchinto** (resource `$statement`, array `&$result` [, int `$mode`])

This function is deprecated. Recommended alternatives: oci_fetch_array(), oci_fetch_object(), oci_fetch_assoc() and oci_fetch_row().

ocifetchstatement

ocifetchstatement -- Alias of oci_fetch_all()

Description

This function is an alias of: oci_fetch_all().

ocifreecollection

ocifreecollection -- Alias of OCI-Collection->free

Description

This function is an alias of: OCI-Collection->free.

ocifreecursor

ocifreecursor -- Alias of oci_free_statement()

Description

This function is an alias of: oci_free_statement().

ocifreedesc

ocifreedesc -- Alias of OCI-Lob->free

Description

This function is an alias of: OCI-Lob->free.

ocifreestatement

ocifreestatement -- Alias of oci_free_statement()

Description

This function is an alias of: oci_free_statement().

ociinternaldebug

ociinternaldebug -- Alias of oci_internal_debug()

Description

This function is an alias of: oci_internal_debug().

ociloadlob

ociloadlob -- Alias of OCI-Lob->load

Description

This function is an alias of: OCI-Lob->load.

ocilogoff

ocilogoff -- Alias of oci_close()

Description

This function is an alias of: oci_close().

ocilogon

ocilogon -- Alias of oci_connect()

Description

This function is an alias of: oci_connect().

ocinewcollection

ocinewcollection -- Alias of oci_new_collection()

Description

This function is an alias of: oci_new_collection().

ocinewcursor

ocinewcursor -- Alias of oci_new_cursor()

Description

This function is an alias of: oci_new_cursor().

ocinewdescriptor

ocinewdescriptor -- Alias of oci_new_descriptor()

Description

This function is an alias of: oci_new_descriptor().

ocinlogon

ocinlogon -- Alias of oci_new_connect()

Description

This function is an alias of: oci_new_connect().

ocinumcols

ocinumcols -- Alias of oci_num_fields()

Description

This function is an alias of: oci_num_fields().

ociparse

ociparse -- Alias of oci_parse()

Description

This function is an alias of: oci_parse().

ociplogon

ociplogon -- Alias of oci_pconnect()

Description

This function is an alias of: oci_pconnect().

ociresult

ociresult -- Alias of oci_result()

Description

This function is an alias of: oci_result().

ocirollback

ocirollback -- Alias of oci_rollback()

Description

This function is an alias of: oci_rollback().

ocirowcount

ocirowcount -- Alias of oci_num_rows()

Description

This function is an alias of: oci_num_rows().

ocisavelob

ocisavelob -- Alias of OCI-Lob->save

Description

This function is an alias of: OCI-Lob->save.

ocisavelobfile

ocisavelobfile -- Alias of OCI-Lob->import

Description

This function is an alias of: OCI-Lob->import.

ociserverversion

ociserverversion -- Alias of oci_server_version()

Description

This function is an alias of: oci_server_version().

ocisetprefetch

ocisetprefetch -- Alias of oci_set_prefetch()

Description

This function is an alias of: oci_set_prefetch().

ocistatementtype

ocistatementtype -- Alias of oci_statement_type()

Description

This function is an alias of: oci_statement_type().

ociwritelobtofile

ociwritelobtofile -- Alias of OCI-Lob->export

Description

This function is an alias of: OCI-Lob->export.

ociwritetemporarylob

ociwritetemporarylob -- Alias of OCI-Lob->writeTemporary

Description

This function is an alias of: OCI-Lob->writeTemporary.