Глава 5. Взаимодействие с базами данных

ODBC

В дополнение к поддержке обычных драйверов ODBC, данная группа функций позволяет работать с базами данных, поддерживающими так называемый интерфейс Unified ODBC. В число БД, использующих API Unified ODBC входят: **Adabas D** (http://www.adabas.com), **IBM DB2** (http://www.ibm.com/db2), **iODBC** (http://www.iodbc.org), **Solid** (http://www.solidtech.com), и **Sybase SQL Anywhere** (http://www.sybase.com).

Схема работы с различными БД практически идентична и различается только в отношении внутреннего функционирования серверов БД.

См. также раздел «Инсталляция в системах Unix» относительно конфигурирования поддержки этих БД при компиляции.

Заметьте, что использование драйверов ODBC снижает производительность, и не является оправданным, если конечно вы не планируете сменять БД вашего приложения достаточно часто. Более обосновано использовать первичные функции PHP интерфейса соответствующей БД.

Но, что касается БД с интерфейсом Unified ODBC, то здесь не используются сами ODBC драйверы, просто все они используют единую архитектуру API.

odbc_connect. Подключение к источнику данных (БД)

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

Возвращает дескриптор подключения к БД (используемый последующими функциями), или 0 (`false`) при ошибке. Одновременно можно открывать несколько подключений. В общем случае для открытия подключения необходимо указывать имя системного источника данных DSN, имя и пароль клиента. В Unix системах иногда возникает проблема подключений, которая может быть устранена указанием параметров подключения в одной строке аргумента `dsn`, в виде «`DSN=DataSource;UID=UserName;PWD=Password`».

В необязательном четвертом аргументе можно указать тип курсора БД, и это часто помогает решить проблемы с некоторыми БД; например, при попытке выполнения хранимых процедур (при вызове которых возникает ошибка типа: «Cannot open a cursor on a stored procedure that has anything other than a single select statement in it»), и в таких случаях может помочь указание типа курсора `SQL_CUR_USE_ODBC`. Тип курсора может быть указан одной из следующих констант:

- `SQL_CUR_USE_IF_NEEDED`
- `SQL_CUR_USE_ODBC`

- SQL_CUR_USE_DRIVER
- SQL_CUR_DEFAULT

odbc_pconnect. Создание устойчивого подключения

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

Возвращает дескриптор подключения к БД, или `false` при ошибке.

Перед подключением, функция пытается проверить имеется ли уже открытое (устойчивое) подключение с параметрами (имя источника данных, пользователя и паролем), аналогичными указанным. Если такое подключение обнаруживается, то возвращается его идентификатор, вместо создания нового подключения.

При завершении скрипта, подключение не закрывается, а остается действительным для дальнейшего использования. Но, устойчивые подключения не работают, если PHP запускается в режиме CGI.

odbc_close. Закрытие сеанса подключения ODBC

```
void odbc_close (int connection_id)
```

Заметьте, при имеющихся незавершенных транзакциях, функция не закрывает подключение.

odbc_close_all. Закрытие всех подключений ODBC

```
void odbc_close_all(void)
```

odbc_cursor. Определение типа курсора для подключения

```
string odbc_cursor (int result_id)
```

Возвращает имя курсора используемого в подключении `result_id`.

odbc_exec. Выполнение запроса SQL

```
int odbc_exec (int connection_id, string query_string)
```

Возвращает дескриптор набора записей, возвращенных запросом `query_string`; или `false` при ошибке. Параметр `connection_id` должен быть ранее успешно возвращен функцией `odbc_connect()` или `odbc_pconnect()`.

См. также: `odbc_prepare()` и `odbc_execute()`.

odbc_prepare. Подготовка запроса

```
int odbc_prepare (int connection_id, string query_string)
```

Функция возвращает дескриптор запроса, который затем используется в функции odbc_execute() для его исполнения; или false в случае ошибки.

odbc_execute. Выполнение подготовленного запроса

int **odbc_execute** (int result_id [, array parameters_array])

Выполняет запрос result_id, подготовленный ранее функцией odbc_prepare(). Возвращает true; или false при ошибке. В массиве parameters_array можно указать параметры запроса, если они требуются.

odbc_autocommit. Переключение режима транзакций autocommit

int **odbc_autocommit** (int connection_id [, int OnOff])

Если не указывается аргумент OnOff, функция возвращает текущее состояние auto-commit для подключения connection_id. True означает «разрешено», а false «не разрешено» или произошла ошибка.

Если в значении аргумента OnOff указывается значение true, то последующие запросы исполняются незамедлительно, если false – то они откладываются до вызова функции odbc_commit() (таким образом формируя транзакцию). При этом возвращается true, или false при ошибке.

По умолчанию все запросы исполняются незамедлительно, и запрещение состояния auto-commit равносильно началу транзакции.

См. также: odbc_commit() и odbc_rollback().

odbc_commit. Завершение транзакции

int **odbc_commit** (int connection_id)

Выполняет все отложенные запросы (транзакцию) для подключения connection_id и возвращает true.

odbc_rollback. Отмена транзакции

int **odbc_rollback** (int connection_id)

Отменяет все отложенные запросы (транзакцию) для подключения connection_id и возвращает true.

odbc_num_rows. Получение числа возвращенных записей

int **odbc_num_rows** (int result_id)

В аргументе указывается дескриптор результата, возвращенный ODBC запросом. При ошибке возвращается –1. Для запросов INSERT, UPDATE и DELETE возвращается число вставленных, измененных, удаленных записей; для запроса

SELECT число возвращенных запросом записей (некоторые драйвера возвращают –1, вне зависимости от того, сколько записей было возвращено).

odbc_num_fields. Число полей в результате

```
int odbc_num_fields (int result_id)
```

В аргументе указывается дескриптор результата, возвращенный ODBC запросом (функцией odbc_exec()). Функция возвращает число полей (столбцов) содержащихся в наборе записей, возвращенных запросом; либо –1 в случае ошибки.

Обработка полей LONGVARBINARY определяется odbc_binmode().

odbc_result. Получение данных результата запроса

```
string odbc_result (int result_id, mixed field)
```

Возвращает содержимое поля указанного аргументом field, текущей записи. Поле может быть указано его номером (начиная с 1) или именем:

```
$item_3 = odbc_result ($Query_ID, 3);
$item_val = odbc_result ($Query_ID, "id");
```

Данные полей типа long или двоичные, возвращаются согласно установкам функций odbc_binmode () и odbc_longreadlen().

odbc_result_all. Распечатка результата запроса в таблице HTML

```
int odbc_result_all (int result_id [, string format])
```

Возвращает число выведенных записей; или false при ошибке. В аргументе result_id указывается дескриптор набора записей, возвращенный функцией odbc_exec(), а в строке format можно указать атрибуты тега "<TABLE ... >".

odbc_fetch_row. Установка курсора текущей записи

```
int odbc_fetch_row (int result_id [, int row_number])
```

Выбирает запись номер row_number (или по умолчанию следующую) из набора записей result_id, возвращенного функциями odbc_do() или odbc_exec(). Возвращает true; или false при ошибке (например, когда записи с таким номером не существует). Последующие вызовы odbc_result() будут возвращать данные из выбранной записи. Нумерация начинается с 0. Не все драйверы допускают произвольный выбор записи.

```
$cnn = odbc_connect("MyDSN","root","pass123");
$rs = odbc_exec($cnn,"SELECT * FROM tbl1");
echo "Записей: ",  $nr = odbc_num_rows($rs);
echo ",\tполей: ", $nf = odbc_num_fields($rs);

// распечатать в виде [n] 'aaa', 'ббб', 'ввв',
for($r=1; $r<=$nr; $r++) {    // записи
```

```
    echo "\n[$r] ";
    odbc_fetch_row($rs,$r);
    for($f=1; $f<=$nf; $f++) // поля
        echo "'", odbc_result($rs,$f), "',";
}
```

odbc_fetch_into. Занесение полученной записи в массив

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

Заносит поля записи номер `rownumber` из набора записей `result_id` в элементы массива `result_array`. Возвращает число полей набора записей (результата запроса); или `false` при ошибке.

```
$cnn = odbc_connect("MyDSN","root","pass123");
$rs = odbc_exec($cnn,"SELECT * FROM tbl1");

for($r=1; $r<=odbc_num_rows($rs); $r++) {
    odbc_fetch_into($rs,$r,$arr);
    $ar[]=$arr;  // занести набор записей в двухмерный массив
}
```

odbc_field_name. Определение имени поля

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

Возвращает имя поля по его номеру `field_number`, в наборе записей `result_id`. Нумерация начинается с единицы. При ошибке возвращает `false`.

odbc_field_num. Определение порядкового номера поля

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

Возвращает номер поля по его имени `field_name`, в наборе записей `result_id`. Нумерация начинается с 1. При ошибке возвращает `false`.

odbc_field_type. Получение типа данных поля

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

Возвращает SQL тип поля с номером `field_number`, в наборе записей `result_id`. Нумерация начинается с 1. При ошибке возвращает `false`.

odbc_field_len. Получение длины (точности) поля

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

См. также: `odbc_field_scale()`.

odbc_field_precision. Синоним odbc_field_len()

```
string odbc_field_precision (int result_id, int field_number)
```

odbc_field_scale. Определение точности поля

```
string odbc_field_scale (int result_id, int field_number)
```

Функция возвращает число сохраняемых десятичных знаков после запятой для полей дробных чисел.

odbc_free_result. Освобождение ресурсов занятых набором записей

```
int odbc_free_result (int result_id)
```

Всегда возвращает `true`. Поскольку при завершении скрипта освобождаются все выделенные ему ресурсы, использовать данную функцию имеет смысл, только в случае крайней экономии памяти, расходуемой скриптом.

Если транзакция была начата, но не завершена (см. `odbc_autocommit()`) данная функция отменяет ее.

odbc_binmode. Определение обработки двоичных полей

```
int odbc_binmode (int result_id, int mode)
```

Функция управляет обработкой полей ODBC SQL типов: `BINARY`, `VARBINARY`, `LONGVARBINARY`. Если `result_id` равен 0, то установки будут применяться к новым наборам записей.

- `ODBC_BINMODE_PASSTHRU` – пропускать двоичные поля
- `ODBC_BINMODE_RETURN` – возвращать как есть (по ум.)
- `ODBC_BINMODE_CONVERT` – возвращать, конвертируя

Пропускание (в первом случае) означает, что при использовании функции `odbc_fetch_into()`, для полей этого типа будет возвращаться пустая строка.

В последнем случае конвертация означает, что каждый байт данных будет представлен двумя шестнадцатеричными цифрами, например двоичный байт `00000001` будет представлен как «01», а `11111111` как «FF».

Обработка длинных двоичных полей также зависит от опции установленной функцией `odbc_longreadlen()`.

Обработка полей LONGVARBINARY

Binmode	longreadlen	**Результат**
`ODBC_BINMODE_PASSTHRU`	0	Пропускать
`ODBC_BINMODE_RETURN`	0	Пропускать
`ODBC_BINMODE_CONVERT`	0	Пропускать
`ODBC_BINMODE_PASSTHRU`	>0	Пропускать

ODBC_BINMODE_RETURN	>0	Возвращать, как есть
ODBC_BINMODE_CONVERT	>0	Конвертировать

odbc_longreadlen. Обработка полей LONG

int **odbc_longreadlen** (int result_id, int length)

Функция управляет обработкой полей ODBC SQL типов: LONG, LONGVARBINARY. В аргументе length указывается, сколько байтов следует возвращать из полей данных типов. По умолчанию, 4096 байт. См. также: odbc_binmode().

odbc_setoption. Настройка опции ODBC

int **odbc_setoption** (int id, int function, int option, int param)

Если в аргументе function указывается значение 1, то тогда настраиваются параметры подключения (в этом случае в аргументе id необходимо указывать дескриптор подключения), если указывается 2, то настраиваются параметры запроса (тогда в id указывается настраиваемый запрос).

Это позволяет устранить проблемы с некоторыми своеобразными драйверами ODBC. Конечно, вы должны понимать какой эффект будет иметь установка той или иной опции для конкретного драйвера. Пользуйтесь документацией ODBC и не забывайте, что функциональность различных версий драйверов может различаться.

Так как использование этой функции сильно зависит от специфики ODBC драйвера, использовать ее в публикуемых скриптах PHP не рекомендуется. Имейте также ввиду, что некоторые опции доступны только после открытия подключения или подготовки запроса.

Аргумент option – номер устанавливаемой опции, а param – ее значение.

При ошибке возвращается false; иначе true.

```
// Номер опции SQL_AUTOCOMMIT в SQLSetConnectOption(): 102
// Значение 1 для SQL_AUTOCOMMIT  - SQL_AUTOCOMMIT_ON.
// Следующая строка имеет тот же эффект, что и
//    odbc_autocommit($conn, true);
odbc_setoption ($conn, 1, 102, 1);

// Номер опции SQL_QUERY_TIMEOUT в SQLSetStmtOption(): 0
// Следующая строка устанавливает предел выполнения запроса в 30 сек.

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

odbc_tables. Получение списка таблиц источника данных

`int` **`odbc_tables`** `(int connection_id [, string qualifier [, string owner [, string name [, string table_type]]]])`

Возвращает в наборе записей перечисление имеющихся в источнике данных `connection_id` таблиц (определенного типа `table_type`). Возвращаемый набор записей содержит следующие поля:

- `TABLE_QUALIFIER`
- `TABLE_OWNER`
- `TABLE_NAME` – имя таблицы
- `TABLE_TYPE` – тип таблицы; например, «TABLE», «VIEW»
- `REMARKS` – примечания; например, «MySQL table»

Возвращаемый набор записей сортируется с порядке: `TABLE_TYPE`, `TABLE_QUALIFIER`, `TABLE_OWNER`, `TABLE_NAME`.

Аргументы `owner` и `name` могут содержать шаблоны ('%' заменяет ноль и более символов, а '_' один символ).

Чтобы выяснить (получить перечисление возможных значений) аргументов qualifier, owner, и table type, используется следующая семантика:

- Если `qualifier` = "%", а `owner` и `name` – пустые строки, то возвращается набор записей, содержащий перечисление возможных квалификаторов источника данных. (Все поля, за исключением `TABLE_QUALIFIER` содержат пустые значения `NULL`.) Обычно это каталоги (БД), но не все БД используют их.
- Если `owner` = "%", а `qualifier` и `name` – пустые строки, то возвращается набор записей содержащий перечисление владельцев источника данных. (Все поля, за исключением `TABLE_OWNER` содержат пустые значения NULL.)
- Если `table_type` = "%", а `qualifier`, `owner` и `name` – пустые строки, то возвращается набор записей, содержащий перечисление возможных типов таблиц источника данных. (Все поля, за исключением `TABLE_TYPE` содержат пустые значения `NULL`.)

Если `table_type` не пустая строка, то она должна содержать перечисление через запятую типы запрашиваемых таблиц, (например в виде: «'TABLE','VIEW'» или «TABLE, VIEW»).

```
$rs = odbc_tables($cnn, "","","%","TABLE" ); // все имеющиеся таблицы
```

См. также: `odbc_tableprivileges()`.

odbc_tableprivileges. Получение списка привилегий таблиц

`int` **`odbc_tableprivileges`** `(int connection_id [, string qualifier [, string owner [, string name]]])`

Возвращает список таблиц и присвоенных им привилегий в наборе записей; или false при ошибке. Возвращаемый набор записей сортируется с порядке: TABLE_QUALIFIER, TABLE_OWNER, TABLE_NAME и содержит следующие поля:

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

Аргументы owner и name могут содержать шаблоны ('%' заменяет ноль и более символов, а '_' один символ).

См. описание: odbc_tables().

odbc_statistics. Получение описания таблицы

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

Возвращает список таблиц и их индексов в наборе записей; или false при ошибке. Возвращаемый набор записей сортируется в порядке: NON_UNIQUE, TYPE, INDEX_QUALIFIER, INDEX_NAME и SEQ_IN_INDEX и содержит следующие поля:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- NON_UNIQUE
- INDEX_QUALIFIER
- INDEX_NAME
- TYPE
- SEQ_IN_INDEX
- COLUMN_NAME
- COLLATION
- CARDINALITY
- PAGES
- FILTER_CONDITION

odbc_columns. Перечисление полей заданной таблицы

`int` **`odbc_columns`** `(int connection_id [, string qualifier [, string owner [, string table_name [, string column_name]]]])`

Возвращает список полей таблицы `table_name` в наборе записей; или `false` при ошибке. Возвращаемый набор записей сортируется в порядке: `TABLE_QUALIFIER`, `TABLE_OWNER` и `TABLE_NAME` и содержит следующие поля:

- `TABLE_QUALIFIER`
- `TABLE_OWNER`
- `TABLE_NAME`
- `COLUMN_NAME`
- `DATA_TYPE`
- `TYPE_NAME`
- `PRECISION`
- `LENGTH`
- `SCALE`
- `RADIX`
- `NULLABLE`
- `REMARKS`

Аргументы `owner`, `table_name` и `column_name` могут содержать шаблоны ('%' заменяет ноль и более символов, а '_' один символ).

См. также: `odbc_columnprivileges()`.

odbc_columnprivileges. Перечисление полей заданной таблицы с их привилегиями

`int` **`odbc_columnprivileges`** `(int connection_id [, string qualifier [, string owner [, string table_name [, string column_name]]]])`

Возвращает список полей таблицы `table_name` в наборе записей; или `false` при ошибке. Возвращаемый набор записей сортируется в порядке: `TABLE_QUALIFIER`, `TABLE_OWNER` и `TABLE_NAME` и содержит следующие поля:

- `TABLE_QUALIFIER`
- `TABLE_OWNER`
- `TABLE_NAME`
- `GRANTOR`
- `GRANTEE`
- `PRIVILEGE`

- IS_GRANTABLE

Аргумент column_name может содержать шаблоны ('%' заменяет ноль и более символов, а '_' один символ).

odbc_gettypeinfo. Определение типов поддерживаемых данных

int **odbc_gettypeinfo** (int connection_id [, int data_type])

Возвращает набор записей содержащий перечисление поддерживаемых БД типов данных, если необходимо выяснить поддерживается ли конкретный тип данных, его необходимо указать в аргументе data_type. При ошибке возвращается false.

В наборе записей возвращаются следующие поля:

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

Записи сортируются по полям DATA_TYPE и TYPE_NAME.

odbc_primarykeys. Подбор поля способного быть первичным ключом таблицы

int **odbc_primarykeys** (int connection_id, string qualifier, string owner, string table)

Возвращает набор записей, содержащий список полей, которые могут претендовать на роль первичного ключа таблицы table. При ошибке возвращается false.

В наборе записей возвращаются следующие поля:

- TABLE_QUALIFIER
- TABLE_OWNER

- TABLE_NAME
- COLUMN_NAME
- KEY_SEQ
- PK_NAME

odbc_foreignkeys. Список внешних ключей

```
int odbc_foreignkeys (int connection_id,
      string pk_qualifier, string pk_owner, string pk_table,
      string fk_qualifier, string fk_owner, string fk_table)
```

Если pk_table содержит имя таблицы, то функция возвращает набор записей, содержащий первичный ключ этой таблицы и все внешние ключи (foreign keys), которые на него ссылаются.

Если fk_table содержит имя таблицы, то функция возвращает набор записей, содержащий все внешние ключи этой таблицы и корреспондирующие первичные ключи в других таблицах.

Если и pk_table и fk_table содержат имена таблиц, функция возвращает внешний ключ таблицы fk_table, который указывает на первичный ключ в таблице pk_table.

В возвращаемом наборе записей содержатся следующие поля:

- PKTABLE_QUALIFIER
- PKTABLE_OWNER
- PKTABLE_NAME
- PKCOLUMN_NAME
- FKTABLE_QUALIFIER
- FKTABLE_OWNER
- FKTABLE_NAME
- FKCOLUMN_NAME
- KEY_SEQ
- UPDATE_RULE
- DELETE_RULE
- FK_NAME
- PK_NAME

odbc_procedures. Список хранимых процедур

```
int odbc_procedures (int connection_id [, string qualifier [,
string owner [, string name]]])
```

Возвращает список хранимых процедур name в наборе записей, содержащем следующие поля:

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

Аргументы owner, и name могут содержать шаблоны ('%' заменяет ноль и более символов, а '_' один символ).

odbc_procedurecolumns. Список параметров хранимых процедур

int **odbc_procedurecolumns** (int connection_id [, string qualifier [, string owner [, string proc [, string column]]]])

Возвращает список хранимых процедур (с их входными и выходными параметрами и описанием возвращаемых ими полей) в наборе записей, содержащем следующие поля:

- PROCEDURE_QUALIFIER
- PROCEDURE_OWNER
- PROCEDURE_NAME
- COLUMN_NAME
- COLUMN_TYPE
- DATA_TYPE
- TYPE_NAME
- PRECISION
- LENGTH
- SCALE
- RADIX
- NULLABLE
- REMARKS

Записи сортируются по полям: PROCEDURE_QUALIFIER, PROCEDURE_OWNER, PROCEDURE_NAME и COLUMN_TYPE.

Аргументы owner, proc и column могут содержать шаблоны ('%' заменяет ноль и более символов, а '_' один символ).

odbc_specialcolumns. Получение списка специальных полей

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

Когда в аргументе type указано значение SQL_BEST_ROWID, возвращаются поля, уникально идентифицирующие каждую запись в таблице.

Когда в аргументе type указано значение SQL_ROWVER, возвращаются оптимальные поля, извлекая значения которых, можно уникально идентифицировать любую запись указанной таблицы (вне зависимости от изменений производимых в таблице).

Поля возвращаются в наборе записей (сортируемом по полю SCOPE), содержащем следующие поля:

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

MySQL

Эти функции позволяют работать с серверами БД MySQL. См. также документацию по адресу www.mysql.com/documentation.

mysql_connect. Подключение к серверу MySQL

int **mysql_connect** ([string hostname[:port][:/path/to/socket] [, string username [, string password]]])

Возвращает дескриптор подключения к БД, или false при ошибке. Обычно, следующим шагом является выбор БД на сервере, функцией mysql_select_db().

Для неуказанных аргументов, используются следующие умолчания: host:port = 'localhost:3306', username = имя пользователя запустившего текущий процесс сервера password = ""; либо значения из файла конфигурации.

Строка hostname может также включать номер порта (в виде «hostname:port»), или путь к сокету для локальной машины в системах Unix ":/path/to/socket».

При ошибке также выдается предупреждение, выдачу которого можно блокировать, указав перед именем функции оператор '@'.

Если функция повторно вызывается с теми же аргументами, новое подключение не создается, а возвращается идентификатор имеющегося.

В конце скрипта принято закрывать подключение функцией `mysql_close()`, но это можно не делать, поскольку PHP автоматически закрывает все (неустойчивые) подключения при завершении скрипта.

```
<?php
    $link = @mysql_connect ("server", "user", "passwd")
        or die ("Невозможно подключиться");
    print ("Подключение к MySQL успешно");
    mysql_close ($link);
?>
```

См. также: `mysql_pconnect()`, и `mysql_close()`.

mysql_pconnect. Создание устойчивого подключение к серверу MySQL

`int` **`mysql_pconnect`** `([string hostname[:port][:/path/to/socket] [, string username [, string password]]])`

Возвращает дескриптор подключения к БД, или `false` при ошибке (также выдается предупреждение). Обычно, следующим шагом является выбор БД на сервере, функцией `mysql_select_db()`.

Для неуказанных аргументов, используются следующие умолчания: `host:port` = 'localhost:3306', `username` = имя пользователя запустившего текущий процесс сервера `password` = ""; либо значения из файла конфигурации.

Строка `hostname` может также включать номер порта (в виде «`hostname:port`»), или путь к сокету для локальной машины в системах Unix "`:/path/to/socket`».

`mysql_pconnect()` действует подобно `mysql_connect()` с двумя различиями:

Перед подключением, функция пытается проверить имеется ли уже открытое (устойчивое) подключение с параметрами (имя сервера, пользователя и пароль), аналогичными указанным. Если такое подключение обнаруживается, то возвращается его идентификатор, вместо создания нового подключения.

При завершении скрипта, подключение не закрывается, а остается действительным для дальнейшего использования. (Функция `mysql_close()` не может закрыть подключения, созданные с помощью `mysql_pconnect()`).

mysql_close. Отключение от сервера MySQL

`int` **`mysql_close`** `([int link_identifier])`

Возвращает `true`, или `false` при ошибке.

Идентификатор закрываемого подключения указывается в аргументе, если его не указывать, то закрывается последнее, открытое данным скриптом подключение.

Фактически использование данной функции не является обязательным, так как PHP автоматически закрывает все незакрытые неустойчивые подключения при завершении скрипта.

Заметьте, устойчивые подключения, созданные функцией `mysql_pconnect()` не закрываются.

См. также: `mysql_connect()`, и `mysql_pconnect()`.

mysql_change_user. Изменение параметров подключения

`int` **`mysql_change_user`** `(string user, string password [, string database [, int link_identifier]])`

Если не указываются БД или подключение, то используется последняя активная. Если авторизация проходит безуспешно, параметры подключения не изменяются.

Работает с MySQL 3.23.3 и выше.

mysql_list_dbs. Получение списка БД на сервере MySQL

`int` **`mysql_list_dbs`** `([int link_identifier])`

Возвращает набор записей, содержащий список БД на сервере.

```
$link = mysql_connect('localhost', 'myname', 'secret');
$db_list = mysql_list_dbs($link);

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

В данном примере также может использоваться функция `mysql_fetch_row()` или ей подобная.

Заметьте, список можно получить, не имея привелегий, то есть не указывая пароль доступа.

Ранее функция ранее называлась `mysql_listdbs()`.

mysql_db_name. Получение имени БД из списка

`int` **`mysql_db_name`** `(int result, int row [, mixed field])`

В аргументе `result` указывается дескриптор набора записей полученный от функции `mysql_list_dbs()`. В аргументе `row` указывается номер записи. При ошибке возвращает `FALSE`. Фактически данная функция является излишней.

```
<?php
error_reporting(E_ALL);
```

```
mysql_connect('dbhost', 'username', 'password');
$db_list = mysql_list_dbs();

for ($i = 0; $i < ($cnt= mysql_num_rows($db_list)); $i++) {
    echo mysql_db_name($db_list, $i) . "\n";
}
?>
```

Ранее функция называлась `mysql_dbname()`.

mysql_select_db. Выбор БД MySQL

`int` **`mysql_select_db`** `(string database_name [, int link_identifier])`

Возвращает `true`, или `false` при ошибке.

Устанавливает БД с именем `database_name` активной для текущего подключения или указанного в `link_identifier`. Если подключений не имеется, то косвенно вызывается функция `mysql_connect()` с параметрами по умолчанию.

Последующие запросы, выполняемые функцией, будут адресованы данной БД.

См. также: `mysql_connect()`, `mysql_pconnect()`, и `mysql_query()`.

Ранее функция называлась `mysql_selectdb()`.

mysql_create_db. Создание БД MySQL

`int` **`mysql_create_db`** `(string dbname [, int link_identifier])`

Создает БД `dbname`, используя подключение `link_identifier`.

```
<?php
    $link = mysql_pconnect ("localhost", "me", "paswd");
    if (mysql_create_db ("my_db")) {
        print ("БД создана \n");
    } else {
        printf ("Ошибка: %s\n", mysql_error ());
    }
?>
```

Ранее функция называлась `mysql_createdb()`. См. также: `mysql_drop_db()`.

mysql_drop_db. Удаление БД

`int` **`mysql_drop_db`** `(string database_name [, int link_identifier])`

Возвращает `true`, если БД `database_name` успешно удалена; или `false` при ошибке.

См. также: mysql_create_db(). Ранее функция называлась mysql_dropdb().

mysql_list_tables. Получение списка таблиц в БД

int **mysql_list_tables** (string database [, int link_identifier])

Возвращает набор записей, из которого можно извлечь имена таблиц функцией mysql_tablename(). Следующий пример распечатывает имена всех БД и таблиц, в них содержащихся.

```
<?
$link = mysql_connect('127.0.0.1', 'root', '157984');
$db_list = mysql_list_dbs($link);

while ($r_db = mysql_fetch_object($db_list)) {
  echo $r_db->Database . "\n";
      // распечатать список таблиц
  $tbl_list = mysql_list_tables($r_db->Database);
  for($i = 0; $i < mysql_num_rows($tbl_list); $i++) {
      echo " - ", mysql_tablename($tbl_list, $i), "\n";
    }
}
?>
```

Ранее функция называлась mysql_listtables().

mysql_tablename. Получение имени таблицы

string **mysql_tablename** (int result, int i)

Функция используется для получения имени таблицы (с номером i) из набора записей, возвращенного функцией mysql_list_tables().

```
<?php
mysql_connect ("localhost:3306");
$result = mysql_list_tables ("mydb");
$i = 0;
while ($i < mysql_num_rows ($result)) {
    $tb_names[$i] = mysql_tablename ($result, $i);
    echo $tb_names[$i] . "<BR>";
    $i++;
}
?>
```

mysql_query. Выполнение запроса к БД

int **mysql_query** (string query [, int link_identifier])

Посылает запрос текущей активной БД, для текущего подключения или указанного в `link_identifier`. Если подключений не имеется, то косвенно вызывается функция `mysql_connect()` с параметрами по умолчанию.

Заметьте SQL выражение, указанное в аргументе `query` не должно оканчиваться точкой с запятой. Если выражение содержит ошибки, или при его выполнении возникают ошибки (например, если текущие привилегии не позволяют выполнить запрос), то функция возвращает `FALSE`.

Если запрос успешно выполнился, то возвращается набор записей (не забывайте, он также может содержать 0 записей), который может быть обработан функциями:

- `mysql_result()` – получить элемент набора записей
- `mysql_fetch_array()` – занести запись в массив
- `mysql_fetch_row()` – занести запись в нумерованный массив
- `mysql_fetch_assoc()` – занести запись в ассоциативный массив
- `mysql_fetch_object()` – занести запись в объект

Чтобы выяснить, сколько записей было возвращено командой `SELECT`, используйте функцию `mysql_num_rows()`; а чтобы выяснить сколько записей было изменено в результате выполнения запросов `DELETE`, `INSERT`, `REPLACE`, или `UPDATE`, используйте функцию `mysql_affected_rows()`.

После обработки результатов запроса, он может быть удален функцией `mysql_free_result()`. Хотя в этом нет необходимости, так как ресурсы автоматически освобождаются при завершении скрипта.

См. также: `mysql_db_query()`, `mysql_select_db()`, и `mysql_connect()`.

mysql_db_query. Выполнение запроса к указанной БД

```
int mysql_db_query (string database, string query [, int link_identifier])
```

Функция эквивалентна последовательному выполнению функций:

```
mysql_select_db (string database [, int link_identifier])
mysql_query (string query [, int link_identifier])
```

См. также: `mysql_db_query()`, `mysql_connect()`. Ранее функция называлась `mysql()`.

mysql_num_rows. Получение числа возвращенных записей

```
int mysql_num_rows (int result)
```

Функция возвращает число записей, возвращенных командами `SELECT`. Чтобы выяснить, сколько записей было изменено в результате выполнения запросов

DELETE, INSERT, REPLACE, или UPDATE, используйте функцию mysql_affected_rows().

```
<?php
   $conn = mysql_connect("hostaddress", "username", "password");
   mysql_select_db("database",$conn);
   $rs = mysql_query("SELECT * FROM Accounts",$conn);
   $N = mysql_num_rows($rs);
   echo "Всего записей: $N";
?>
```

См. также: mysql_db_query(), mysql_query() и, mysql_fetch_row().

Ранее функция называлась mysql_numrows().

mysql_affected_rows. Получение числа измененных записей в БД

int **mysql_affected_rows** ([int link_identifier])

Возвращается число записей, измененных в результате выполнения запросов DELETE, INSERT, REPLACE, или UPDATE.

Если последним запросом была команда DELETE без ограничения WHERE, то из таблицы будут удалены все записи, но эта функция возвратит 0.

Чтобы выяснить, сколько записей было возвращено командой SELECT, используйте функцию mysql_num_rows().

mysql_insert_id. Получение значения последнего автоинкремента

int **mysql_insert_id** ([int link_identifier])

При добавлении записей в таблицу командой INSERT, для поля, имеющего свойство AUTO_INCREMENT значение генерируется автоматически (вне зависимости от того было ли оно задано) и его можно получить с помощью это функции. Это значение можно использовать для модификации последней добавленной записи.

Это значение также можно получить с помощью SQL запроса MySQL: «SELECT LAST_INSERT_ID()".

Если последним запросом значение AUTO_INCREMENT сгенерировано не было, то функция возвратит 0. Заметьте также, что если тип автоинкрементируемого поля был BIGINT, то он будет преобразован к типу LONG, и результат может быть неверным.

Команда LAST_INSERT_ID() имеет два преимущества, она всегда хранит значение последнего автоинкремента (это значение не сбрасывается в 0 между запросами) и она всегда возвращает правильное значение.

mysql_data_seek. Перемещение курсора набора записей

int **mysql_data_seek** (int result_identifier, int row_number)

При каждом вызове функции mysql_fetch_row() (или подобной), внутренний курсор записи смещается на следующую. Данная функция позволяет свободно перемещать курсор в наборе записей result_identifier, так, чтобы он указывал на запись с номером row_number (нумерация начинается с 0).

Она возвращает true, или false при ошибке.

```
<?php
    $link = mysql_pconnect ("сервер", "имя", "пароль")
        or die ("Нету коннекта");

    mysql_select_db ("samp_db") or die ("Базу не выбрать");

    $query = "SELECT last_name, first_name FROM friends";
    $result = mysql_query ($query) or die ("Query failed");

    # отобразить записи в обратном порядке

    for ($i = mysql_num_rows ($result) - 1; $i >=0; $i--) {
        if (!mysql_data_seek ($result, $i)) {
            printf ("Cannot seek to row %d\n", $i);
            continue;
        }
        if(!($row = mysql_fetch_object ($result)))
            continue;
        printf ("%s %s<BR>\n", $row->last_name, $row->first_name);
    }

    mysql_free_result ($result);
?>
```

mysql_free_result. Уничтожение набора записей

int **mysql_free_result** (int result)

Функция освобождает память, занимаемую набором записей result, возвращенным запросом. Ее следует использовать только, если очень требуется экономить память, так как память автоматически освобождается при завершении скрипта.

Ранее функция называлась mysql_freeresult().

Обработка результатов запроса

mysql_result. Получение определенного элемента набора записей

mixed **mysql_result** (int result, int row [, mixed field])

Возвращает содержимое ячейки из набора записей result. В аргументе row указывается номер записи (нумерация начинается с 0), в аргументе field можно указать индекс поля (число), имя поля или полное имя поля (вида: «имя_таблицы.имя_поля») или алиас поля (для запросов типа 'SELECT foo AS bar FROM...').

При работе с большими наборами записей более быстро выполняются функции типа _fetch_ (см. ниже). Также заметьте, что численные индексы обрабатываются быстрее строковых.

Не следует вызывать функцию mysql_result() в сочетании с другими функциями обработки набора записей.

Рекомендуется использовать альтернативные функции: mysql_fetch_row(), mysql_fetch_array(), и mysql_fetch_object().

mysql_fetch_array. Занесение записи в массив

array **mysql_fetch_array** (int result [, int result_type])

Возвращает массив, соответствующий текущей записи, из набора записей result, возвращенных запросом; или false записей более не имеется.

Данная функция является расширением функции mysql_fetch_row(), и она может возвращать нумерованный или ассоциативный массив (или объединенный). Вид возвращаемого массива может указываться в аргументе result_type одной из констант: MYSQL_NUM, MYSQL_ASSOC, MYSQL_BOTH (по умолчанию).

В ассоциативных массивах индексами служат имена полей. Если имеются одноименные поля, используется последнее. Для доступа к одноименным полям также можно использовать числовые индексы (поля нумеруются в той последовательности, в которой указаны в запросе или в таблице) или алиасы.

```
SELECT  t1.f1 AS t1_f1  t2.f1 AS t2_f1  FROM t1, t2
```

Заметьте, что функция не значительно медленнее, чем mysql_fetch_row(), но предоставляет дополнительную функциональность.

```
<?php
   mysql_connect ($host, $user, $password);
   $result = mysql_db_query ("database",
                    "select user_id, fullname from tbl");
   while ($row = mysql_fetch_array ($result)) {
    echo "user_id: ".$row["user_id"]."<br>\n";
    echo "user_id: ".$row[0]."<br>\n";
    echo "fullname: ".$row["fullname"]."<br>\n";
    echo "fullname: ".$row[1]."<br>\n";
   }
   mysql_free_result ($result);
?>
```

См. также: mysql_fetch_row() и mysql_fetch_assoc().

mysql_fetch_row. Занесение записи в нумерованный массив

`array` **`mysql_fetch_row`** `(int result)`

Возвращает массив, соответствующий текущей записи, из набора записей `result`, возвращенных запросом (последующий вызов функции возвращает следующую запись); или `false` записей более не имеется.

Каждое поле записи сохраняется в нумерованном элементе массива, (нумерация начинается с 0).

См. также: `mysql_fetch_array()`, `mysql_fetch_object()`, `mysql_data_seek()`, `mysql_fetch_lengths()`, и `mysql_result()`.

mysql_fetch_assoc. Занесение записи в ассоциативный массив

`array` **`mysql_fetch_assoc`** `(int result)`

Возвращает массив, соответствующий текущей записи, из набора записей `result`, возвращенных запросом; или `false` записей более не имеется. Вызов функции эквивалентен `mysql_fetch_array(result, MYSQL_ASSOC)`.

В возвращаемом массиве индексами служат имена полей. Если имеются одноименные поля, используется последнее. Для доступа к одноименным полям можно использовать алиасы или функцию `mysql_fetch_array()`.

If two or more columns of the result have the same field names, the last column will take precedence. To access the other column(s) of the same name, you must use and have it return the numeric indices as well.

Заметьте, что функция не значительно медленнее, чем `mysql_fetch_row()`, но предоставляет дополнительную функциональность.

```
<?php
   mysql_connect ($host, $user, $password);
   $result = mysql_db_query ("database","select * from table");
   while ($row = mysql_fetch_assoc ($result)) {
    echo $row["user_id"];
    echo $row["fullname"];
   }
   mysql_free_result ($result);
?>
```

См. также: `mysql_fetch_row()` и `mysql_fetch_array()`.

mysql_fetch_object. Получение записи в свойствах объекта

`object` **`mysql_fetch_object`** `(int result)`

Возвращает объект, в свойствах которого находятся поля текущей записи; или `false` записей более не имеется.

По скорости функция идентична `mysql_fetch_array()`, и почти идентична `mysql_fetch_row()`.

```
<?php
 mysql_connect ($host, $user, $password);
 $result = mysql_db_query ("database", "select * from table");
 while ($row = mysql_fetch_object ($result)) {
    echo $row->user_id;
    echo $row->fullname;
 }
 mysql_free_result ($result);
?>
```

См. также: `mysql_fetch_array()` и `mysql_fetch_row()`.

mysql_fetch_lengths. Получение длины элементов записи

`array` **`mysql_fetch_lengths`** `(int result)`

После того, как запись была обработана одной из функций: `mysql_fetch_row()`, `mysql_fetch_array()`, или `mysql_fetch_object()`, с помощью данной функции можно получить размер полученного значения в каждом обработанном поле.

Например, в следующем фрагменте:

```
$rs_arr = mysql_fetch_row($rs);
$rs_len = mysql_fetch_lengths($rs);
```

массив `$rs_len` будет содержать длину соответствующих элементов массива `$rs_arr`, то есть `$rs_len[0] = strlen($rs_arr[0])`, т. д.

См. также: `mysql_fetch_row()`.

mysql_fetch_field. Получение информации о поле записи в свойствах объекта

`object` **`mysql_fetch_field`** `(int result [, int field_offset])`

Если номер поля `field_offset` не указан, при каждом вызове функции возвращаются свойства следующего поля из набора записей `result`.

Возвращаемый объект имеет следующие свойства (и содержит информацию):

- `name` – имя поля
- `table` – имя таблицы, которой принадлежит поле
- `max_length` – максимальная длинна поля
- `not_null` – 1, если полю разрешено пустое значение
- `primary_key` – 1, если поле является ключевым (primary key)
- `unique_key` – 1, если в поле допускаются только уникальные значения (unique key)
- `multiple_key` – 1, если в поле допустимо иметь повторяющиеся значения

- `numeric` – 1, если поле числовое
- `blob` – 1, если поле имеет тип BLOB
- `type` – тип поля
- `unsigned` – 1, если поле числовое беззнаковое
- `zerofill` – 1, если поле заполняется нулями (zero-filled)

```
<?php
mysql_connect ($host, $user, $password) or die ("Could not connect");
$result = mysql_db_query ("database", "select * from table")
    or die ("Query failed");

for($i = 0; $i < mysql_num_fields ($result); $i++) {
    echo "Метаданные поля $i:<BR>\n";
    $meta = mysql_fetch_field ($result);
    if (!$meta) {
        echo "No information available<BR>\n";
    }
    echo "<PRE>
name:          $meta->name
table:         $meta->table
type:          $meta->type
max_length:    $meta->max_length
not_null:      $meta->not_null
zerofill:      $meta->zerofill
unique_key:    $meta->unique_key
primary_key:   $meta->primary_key
multiple_key:  $meta->multiple_key
numeric:       $meta->numeric
unsigned:      $meta->unsigned
blob:          $meta->blob
</PRE>";
}
mysql_free_result ($result);
?>
```

См. также: `mysql_field_seek()`.

mysql_field_seek. Перемещение к указанному полю

`int` **`mysql_field_seek`** `(int result, int field_offset)`

Функция является излишней и следующие фрагменты эквивалентны:

```
$meta = mysql_fetch_field ($result, field_offset);
```

и

```
mysql_field_seek ($result, field_offset)
```

```
$meta = mysql_fetch_field ($result);
```

Так как эта функция влияет только на `mysql_fetch_field()`, а остальные функции этого рода (см. ниже) требуют явного указания номера поля.

См. также: `mysql_fetch_field()`.

mysql_field_name. Получение имени поля в наборе записей

`string` **`mysql_field_name`** `(int result, int field_index)`

Функция возвращает имя поля с индексом `field_offset` (нумерация начинается с 0) в наборе записей `result`.

```
$res = mysql_db_query("users", "SELECT user_id, username FROM users");
echo mysql_field_name($res, 1); // выведет: username
```

Ранее функция называлась `mysql_fieldname()`.

mysql_field_table. Получение имени таблицы, которой принадлежит поле из набора записей

`string` **`mysql_field_table`** `(int result, int field_offset)`

Ранее функция называлась `mysql_fieldtable()`.

mysql_field_len. Получение размера поля набора записей

`int` **`mysql_field_len`** `(int result, int field_offset)`

Ранее функция называлась `mysql_fieldlen()`.

mysql_field_type. Получение типа поля набора записей

`string` **`mysql_field_type`** `(int result, int field_offset)`

Возвращаемая строка содержит название типа поля: «int», «real», «string», «blob», или другого, описанного в документации по MySQL.

```
<?php

mysql_connect ("localhost:3306");
mysql_select_db ("mydb1");
$result = mysql_query ("SELECT * FROM tbl1");
$fields = mysql_num_fields ($result);
$rows   = mysql_num_rows ($result);
$i = 0;
$table = mysql_field_table ($result, $i);
echo "Таблица '$table' имеет $fields полей и $rows записей<BR>";
echo "Структура таблицы: <BR>";
while ($i < $fields) {
    $type  = mysql_field_type  ($result, $i);
```

```
    $name  = mysql_field_name  ($result, $i);
    $len   = mysql_field_len   ($result, $i);
    $flags = mysql_field_flags ($result, $i);
    echo $type." ".$name." ".$len." ".$flags."<BR>";
    $i++;
}
mysql_close();

?>
```

Ранее функция называлась `mysql_fieldtype()`.

mysql_field_flags. Получение флагов поля записи

`string` **`mysql_field_flags`** `(int result, int field_offset)`

Поля записей в MySQL могут иметь следующие свойства флаги: «`not_null`», «`primary_key`», «`unique_key`», «`multiple_key`», «`blob`», «`unsigned`», «`zerofill`», «`binary`», «`enum`», «`auto_increment`», «`timestamp`».

Функция возвращает перечисление через пробел флагов, имеющихся у поля с индексом `field_offset` (нумерация начинается с 0) в наборе записей `result`. (разделить полученную строку на составляющие можно функцией `explode()`).

Ранее функция называлась `mysql_fieldflags()`.

mysql_list_fields. Получение списка полей таблицы

`int` **`mysql_list_fields`** `(string database_name, string table_name [, int link_identifier])`

Функция возвращает пустой набор записей таблицы `table_name` из БД `database_name`, который можно использовать для получения информации о всех полях, имеющихся в таблице, с помощью функций: `mysql_fetch_field()` `mysql_field_flags()`, `mysql_field_len()`, `mysql_field_name()`, и `mysql_field_type()`.

Заметьте, при ошибке возвращается –1, а также в переменной `$phperrmsg` сохраняется сообщение об ошибке, и (если функция не была вызвана с оператором @, то) распчатывается сообщение об ошибке.

```
$link = mysql_connect('localhost', 'myname', 'secret');

$fields = mysql_list_fields("database1", "table1", $link);
$columns = mysql_num_fields($fields); // число полей в таблице

// распечатать имена всех полей таблицы
for ($i = 0; $i < $columns; $i++) {
  echo mysql_field_name($fields, $i) . "\n";;
}
```

Ранее функция называлась `mysql_listfields()`.

mysql_num_fields. Получение числа полей в наборе записей

`int` **`mysql_num_fields`** `(int result)`

См. также: `mysql_db_query()`, `mysql_query()`, `mysql_fetch_field()`, `mysql_num_rows()`.

Ранее функция называлась `mysql_numfields()`.

mysql_errno. Получение кода ошибки MySQL

`int` **`mysql_er`**`rno ([int link_identifier])`

Возвращает номер ошибки, произошедшей в ходе выполнения последней функции MySQL, или 0, если ошибка не произошла.

Ранее ошибки, происходящие при операциях с MySQL, выдавались в виде предупреждений, но сейчас возникновение ошибок нужно выявлять самостоятельно.

Предупреждения выводятся только при неудачных подключениях. Или при попытке использования несуществующего ресурса (например, набора записей).

```
<?php
   mysql_connect();
 mysql_select_db("non_existent_db");
   echo mysql_errno().": ".mysql_error()."<BR>";

 mysql_query("SELECT * FROM non_existent_table");
   echo mysql_errno().": ".mysql_error()."<BR>";
?>
```

См. также: `mysql_error()`

mysql_error. Получение сообщения об ошибке MySQL

`string` **`mysql_error`** `([int link_identifier])`

Возвращает сообщение об ошибке, произошедшей в ходе выполнения последней функции MySQL, или пустую строку, если ошибка не произошла.

```
<?php
 mysql_connect();
 mysql_select_db("non_existent_db");
 echo mysql_errno().": ".mysql_error()."<BR>";

 $conn = mysql_query("SELECT * FROM non_existent_table");
 echo mysql_errno().": ".mysql_error()."<BR>";
?>
```

См. также: `mysql_errno()`

mSQL

Данная группа функций позволяет работать с серверами БД mSQL. Для того, чтобы их можно было использовать, PHP нужно скомпилировать с опцией --with-msql[=dir], где можно указать директорию размещения mSQL (по умолчанию, это /usr/local/Hughes).

Документацию по mSQL можно найти по адресу www.hughes.com.au/.

msql_connect. Подключение к серверу mSQL

`int` **`msql_connect`** `([string hostname [, string hostname[:port] [, string username [, string password]]]])`

Возвращает дескриптор подключения к БД, или `false` при ошибке.

Если не указывается имя хоста, то производится локальное подключение.

Если функция повторно вызывается с теми же аргументами, новое подключение не создается, а возвращается идентификатор уже имеющегося.

В конце скрипта принято закрывать подключение функцией `msql_close()`, но это можно не делать, поскольку PHP автоматически закрывает все (неустойчивые) подключения при завершении скрипта.

См. также: `msql_pconnect()`, `msql_close()`.

msql_pconnect. Создание устойчивого подключения к серверу mSQL

`int` **`msql_pconnect`** `([string hostname [, string hostname[:port] [, string username [, string password]]]])`

Возвращает дескриптор подключения к БД, или `false` при ошибке.

`msql_pconnect()` действует подобно `msql_connect()` с двумя различиями:

Перед подключением, функция пытается проверить имеется ли уже открытое (устойчивое) подключение с параметрами (имя сервера, пользователя и пароль), аналогичными указанным. Если такое подключение обнаруживается, то возвращается его идентификатор, вместо создания нового подключения.

При завершении скрипта, подключение не закрывается, а остается действительным для дальнейшего использования. (Функция `msql_close()` не может закрыть подключения, созданные с помощью `msql_pconnect()`).

msql_close. Отключение от сервера mSQL

`int` **`msql_close`** `(int link_identifier)`

Возвращает `true`, или `false` при ошибке.

Идентификатор закрываемого подключения указывается в аргументе `link_identifier`, если его не указывать, то закрывается последнее, открытое данным скриптом подключение.

Фактически использование данной функции не является обязательным, так как PHP автоматически закрывает все незакрытые неустойчивые подключения при завершении скрипта.

Заметьте, устойчивые подключения, созданные функцией `msql_pconnect()` не закрываются.

См. также: `msql_connect()` и `msql_pconnect()`.

msql_list_dbs. Получение списка БД на сервере mSQL

```
int msql_list_dbs(void);
```

Возвращает набор записей, содержащий список имеющихся БД на сервере. Используйте функцию `msql_dbname()`, чтобы получить элементы этого списка.

msql_dbname. Получение имени БД mSQL

```
string msql_dbname (int query_identifier, int i)
```

Функция используется для обработки набора записей `query_identifier`, возвращенного `msql_listdbs()`, в аргументе `i` указывается номер записи. Возвращается имя БД, или `FALSE` при ошибке.

Число записей можно определить функцией `msql_numrows()`.

msql_create_db. Создание БД mSQL

```
int msql_create_db (string database name [, int link_identifier])
```

Создает БД `dbname`, используя подключение `link_identifier`.

Функция имеет синоним `msql_createdb()`.

См. также: `msql_drop_db()`.

msql_drop_db. Удаление БД mSQL

```
int msql_drop_db (string database_name, int link_identifier)
```

Возвращает `true`, если БД `database_name` успешно удалена; или `false` при ошибке.

Функция имеет синоним `msql_dropdb()`.

См. также: `msql_create_db()`.

msql_list_tables. Получение списка таблиц в БД

`int` **`msql_list_tables`** `(string database)`

Возвращает набор записей, содержащий список БД на сервере. Для извлечения отдельных записей используется функция `msql_tablename()`.

msql_tablename. Получение имени таблицы

`string` **`msql_tablename`** `(int query_identifier, int i)`

Функция используется для получения имени таблицы (с номером `i`) из набора записей `query_identifier`, возвращенного функцией `mysql_list_tables()`.

```
<?php
 msql_connect ("localhost");
 $result = msql_list_tables ("myDB");
 $i = 0;
 while ($i < msql_numrows ($result)) {
    $tb_names[$i] = msql_tablename ($result, $i);
    echo $tb_names[$i] . "<BR>";
    $i++;
 }
?>
```

msql_select_db. Выбор БД mSQL

`int` **`msql_select_db`** `(string database_name, int link_identifier)`

Возвращает `true`, или `false` при ошибке.

Устанавливает БД с именем `database_name` активной для текущего подключения или указанного в `link_identifier`. Если подключений не имеется, то косвенно вызывается функция `mysql_connect()` с параметрами по умолчанию.

Последующие запросы, выполняемые функцией, будут адресованы данной БД.

Функция имеет синоним `msql_selectdb()`.

См. также: `msql_connect()`, `msql_pconnect()`, и `msql_query()`.

msql_regcase. Создание регулярного выражение для поиска, нечувствительного к регистру

См. также: `sql_regcase()`.

msql_query. Выполнение запроса к БД

`int` **`msql_query`** `(string query, [int link_identifier])`

Посылает запрос текущей активной БД, для текущего подключения или указанного в `link_identifier`. Если подключений не имеется, то косвенно вызывается функция `msql_connect()` с параметрами по умолчанию.

Если выражение содержит ошибки, или при его выполнении возникают ошибки (например, если текущие привилегии не позволяют выполнить запрос), то функция возвращает `FALSE`.

Если запрос успешно выполнился, то возвращается набор записей (не забывайте, он также может содержать 0 записей), который может быть обработан функциями:

- `msql_result()` – получить элемент набора записей
- `msql_fetch_array()` – занести запись в массив
- `msql_fetch_row()` – занести запись в нумерованный массив
- `msql_fetch_object()` – занести запись в объект

Чтобы выяснить, сколько записей было возвращено командой `SELECT`, используйте функцию `msql_num_rows()`; а чтобы выяснить сколько записей было изменено в результате выполнения запросов `DELETE`, `INSERT`, `REPLACE`, или `UPDATE`, используйте функцию `msql_affected_rows()`.

После обработки результатов запроса, он может быть удален функцией `mysql_free_result()`.

См. также: `msql()`, `msql_select_db()`, и `msql_connect()`.

msql. Выполнение запроса к указанной БД

`int` **`msql`** `(string database, string query, int link_identifier)`

Функция эквивалентна `msql_query()`, с тем отличием, что БД `database`, которой посылается запрос, здесь указывается явно.

msql_num_rows. Получение числа возвращенных записей

`int` **`msql_num_rows`** `(int query_identifier)`

Функция возвращает число записей, возвращенных запросом.

См. также: `msql()`, `msql_query()`, и `msql_fetch_row()`.

msql_affected_rows. Получение числа измененных записей в БД

`int` **`msql_affected_rows`** `(int query_identifier)`

Функция возвращает число записей, которых «коснулся» запрос (при выборке, удалении, обновлении).

См. также: `msql_query()`.

msql_result. Получение элемента набора записей

`int` **`msql_result`** `(int query_identifier, int row, mixed field)`

Возвращает содержимое ячейки из набора записей `query_identifier`. В аргументе `row` указывается номер записи (нумерация начинается с 0), в аргументе `field` можно указать индекс поля (число), имя поля или полное имя поля (вида: «`имя_таблицы.имя_поля`») или алиас поля.

При работе с большими наборами записей более быстро выполняются функции типа `_fetch_` (см. ниже). Также заметьте, что численные индексы обрабатываются быстрее строковых.

Не следует вызывать функцию `msql_result()` в сочетании с другими функциями обработки набора записей.

Рекомендуется использовать альтернативные функции: `msql_fetch_row()`, `msql_fetch_array()`, и `msql_fetch_object()`.

msql_data_seek. Перемещение курсора набора записей

`int` **`msql_data_seek`** `(int query_identifier, int row_number)`

При каждом вызове функции `msql_fetch_row()` (или подобной), внутренний курсор записи смещается на следующую. Данная функция позволяет свободно перемещать курсор в наборе записей `query_identifier`, так, чтобы он указывал на запись с номером `row_number` (нумерация начинается с 0).

Она возвращает `true`, или `false` при ошибке.

См. также: `msql_fetch_row()`.

msql_fetch_array. Занесение записи в массив

`int` **`msql_fetch_array`** `(int query_identifier [, int result_type])`

Возвращает массив, соответствующий текущей записи, из набора записей `query_identifier`, возвращенных запросом; или `false` записей более не имеется.

Данная функция является расширением функции `msql_fetch_row()`, и она может возвращать нумерованный или ассоциативный массив (или объединенный). Вид возвращаемого массива может указываться в аргументе `result_type` одной из констант: `MYSQL_NUM`, `MYSQL_ASSOC`, `MYSQL_BOTH` (по умолчанию).

В ассоциативных массивах индексами служат имена полей. Если имеются одноименные поля, используется последнее. Для доступа к одноименным полям также можно использовать числовые индексы (поля нумеруются в той последовательности, в которой указаны в запросе или в таблице) или алиасы.

Заметьте, что функция не значительно медленнее, чем `mysql_fetch_row()`, но предоставляет дополнительную функциональность.

Будьте внимательны с обработкой записей содержащих единственное поле, имеющее значение 0 (или пустую строку, или NULL).

См. также: `msql_fetch_row()`.

msql_fetch_row. Занесение записи в нумерованный массив

`array` **`msql_fetch_row`** `(int query_identifier)`

Возвращает массив, соответствующий текущей записи, из набора записей `query_identifier`, возвращенных запросом (последующий вызов функции возвращает следующую запись); или `false` записей более не имеется.

Каждое поле записи сохраняется в нумерованном элементе массива, (нумерация начинается с 0).

См. также: `msql_fetch_array()`, `msql_fetch_object()`, `msql_data_seek()`, и `msql_result()`.

msql_fetch_object. Получение записи в свойствах объекта

`int` **`msql_fetch_object`** `(int query_identifier [, int result_type])`

Возвращает объект, в свойствах которого находятся поля текущей записи; или `false` записей более не имеется.

По скорости функция идентична `mysql_fetch_array()`, и почти идентична `mysql_fetch_row()`.

См. также: `msql_fetch_array()` и `msql_fetch_row()`.

msql_fetch_field. Получение информации о поле записи в свойствах объекта

`object` **`msql_fetch_field`** `(int query_identifier, int field_offset)`

Если номер поля `field_offset` не указан, при каждом вызове функции возвращаются свойства следующего поля из набора записей `query_identifier`.

Возвращаемый объект имеет следующие свойства (и содержит информацию):

- `name` – имя поля
- `table` – имя таблицы, которой принадлежит поле
- `not_null` – 1, если полю разрешено пустое значение
- `primary_key` – 1, если поле является ключевым
- `unique_key` – 1, если в поле допускаются только уникальные значения
- `type` – тип поля

См. также: `msql_field_seek()`.

msql_field_seek. Перемещение к указанному полю

`int` **`msql_field_seek`** `(int query_identifier, int field_offset)`

Последующий вызов `msql_fetch_field()` (если в нем не указан номер поля) будет возвращать информацию о поле с указанным номером `field_offset`.

См. также: `msql_fetch_field()`.

msql_fieldname. Получение имени поля в наборе записей

`string` **`msql_fieldname`** `(int query_identifier, int field)`

Функция возвращает имя поля с индексом `field` в наборе записей `query_identifier`.

msql_fieldtable. Получение имени таблицы, которой принадлежит поле из набора записей

`int` **`msql_fieldtable`** `(int query_identifier, int field)`

msql_fieldtype. Получение типа поля набора записей

`string` **`msql_fieldtype`** `(int query_identifier, int i)`

Возвращаемая строка содержит название типа поля: «int», «real», «string», «blob», или другого, описанного в документации.

msql_fieldflags. Получение флага поля записи

`string` **`msql_fieldflags`** `(int query_identifier, int i)`

Поля записей в mSQL могут иметь два флага: «`not_null`», «`primary_key`».

Функция возвращает перечисление через пробел флагов, имеющихся у поля с индексом `i` (нумерация начинается с 0) в наборе записей `query_identifier`. (разделить полученную строку на составляющие можно функцией `explode()`).

Если поле флагов не имеет, возвращается пустая строка.

msql_fieldlen. Получение размера поля набора записей

`int` **`msql_fieldlen`** `(int query_identifier, int i)`

msql_free_result. Уничтожение набора записей

`int` **`msql_free_result`** `(int query_identifier)`

Функция освобождает память, занимаемую набором записей `query_identifier`, возвращенным запросом.

msql_list_fields. Получение перечисления полей в результате запроса

`int` **`msql_list_fields`** `(string database, string tablename)`

Функция возвращает пустой набор записей таблицы `tablename` из БД `database`, который можно использовать для получения информации о всех полях, имеющихся в таблице, с помощью функций: `msql_fetch_field()` `msql_field_flags()`, `mysql_field_len()`, `mysql_field_name()`, и `mysql_field_type()`.

Заметьте, при ошибке возвращается –1, а также в переменной `$phperrmsg` сохраняется сообщение об ошибке, и (если функция не была вызвана с оператором @, то) распечатывается сообщение об ошибке.

См. также: `msql_error()`.

msql_num_fields. Получение числа полей в наборе записей

`int` **`msql_num_fields`** `(int query_identifier)`

См. также: `msql()`, `msql_query()`, `msql_fetch_field()`, и `msql_num_rows()`.

Ранее функция называлась `msql_numfields()`.

msql_error. Получение сообщения об ошибке последней функции msql

`string` **`msql_error`** `()`

Возвращает сообщение об ошибке, произошедшей в ходе выполнения последней функции mSQL, или пустую строку, если ошибка не произошла. Ранее ошибки, происходящие при операциях с MySQL, выдавались в виде предупреждений, но сейчас возникновение ошибок нужно выявлять самостоятельно.

PostgreSQL

Postgres, изначально разработанная в UC Berkeley Computer Science Department, явилась одной из первых БД использующих объектно-реляционные принципы, доступные теперь в некоторых коммерческих БД. Она поддерживает язык SQL92/SQL3, транзакции, и расширение типов. PostgreSQL распространяется бесплатно и с открытым кодом. См.: http://www.postgresql.org/.

Начиная с версии 6.3 (03/02/1998) PostgreSQL использует сокеты unix-доменов (адрес сокета записывается в файл /tmp/.s.PGSQL.5432). Чтобы к серверу можно было подключиться через TCP/IP, postmaster необходимо запускать с ключом '-i' (дословно означающим: «listen on TCP/IP sockets as well as Unix domain sockets»).

Ранее в функции подключения параметры указывались в отдельных аргументах, но сейчас они заключаются в одну строку:

```
$conn = pg_Connect("host=myHost port=myPort tty=myTTY options=myOptions
   user=myUser password=myPassword dbname=myDB");
```

Для использования функций интерфейса больших объектов их необходимо помещать внутрь блоков транзакций. Блок транзакции начинается с команды `begin` и завершается командами `commit` или `end`. Отмена транзакций выполняется командами `rollback` или `abort`.

```
<?php
    $database = pg_Connect ("dbname=jacarta");
    pg_exec ($database, "begin");
       $oid = pg_locreate ($database);
       echo ("$oid\n");
       $handle = pg_loopen ($database, $oid, "w");
       echo ("$handle\n");
     pg_lowrite ($handle, "gaga");
     pg_loclose ($handle);
    pg_exec ($database, "commit");
?>
```

pg_connect. Подключение к серверу PostgreSQL

`int` **`pg_connect`** `(string conn_string)`

Возвращает дескриптор подключения к БД (используемый последующими функциями), или `false` при ошибке. В аргументе указываются опции подключения: `host`, `port`, `tty`, `options`, `user`, `password`, `dbname`.

```
<?php
   $dbconn = pg_Connect ("dbname=mary");
   $dbconn2 = pg_Connect ("host=localhost port=5432 dbname=mary");
   $dbconn3 = pg_Connect ("user=mb password=baaaa dbname=mary ");
?>
```

Устаревший синтаксис:

```
$conn = pg_connect ("host", "port", "options", "tty", "dbname").
```

См. также: `pg_pconnect()`.

pg_close. Закрытие подключения

`bool` **`pg_close`** `(int connection)`

Закрывает указанное подключение и возвращает `true` (или `false` при ошибке).

Фактически использование данной функции не является обязательным, так как PHP автоматически закрывает все незакрытые неустойчивые подключения при завершении скрипта.

Заметьте, устойчивые подключения, созданные функцией `pg_pconnect()` не закрываются.

pg_pconnect. Создание устойчивого подключения к серверу PostgreSQL

`int` **`pg_pconnect`** `(string conn_string)`

Возвращает дескриптор устойчивого подключения к БД (используемый последующими функциями), или `false` при ошибке. В аргументе указываются опции подключения: `host`, `port`, `tty`, `options`, `user`, `password`, `dbname`.

Заметьте, устойчивые подключения не закрываются при завершении скрипта и остаются действительными пока PHP остается в памяти (так, что при повторной попытке создать идентичное устойчивое подключение, используется уже имеющееся).

Устаревший синтаксис:

```
$conn = pg_pconnect ("host", "port", "options", "tty", "dbname").
```

См. также: `pg_connect()`.

pg_host. Получение имени сервера к которому осуществлено подключение

`string` **`pg_host`** `(int connection_id)`

Возвращает значение опции `host`, указанной при подключении.

pg_port. Получение номера порта используемого подключением

`int` **`pg_port`** `(int connection_id)`

Возвращает значение опции `port`, указанной при подключении (или «5432»).

pg_tty. Получение имени терминала TTY назначенного подключению

`string` **`pg_tty`** `(int connection_id)`

Возвращает имя потока вывода в который сервер записывает отладочную информацию для указанного подключения.

pg_options. Получение опции подключения

`string` **`pg_options`** `(int connection_id)`

pg_dbname. Получение имени используемой БД

`string` **`pg_dbname`** `(int connection)`

При ошибке возвращает `false`.

pg_set_client_encoding. Установка кодировки используемой клиентом

`int` **`pg_set_client_encoding`** `([int connection, string encoding])`

Возвращает 0, или -1 при ошибке. `encoding` может принимать значения:

SQL_ASCII, EUC_JP, EUC_CN, EUC_KR, EUC_TW, UNICODE, MULE_INTERNAL, LATINX (X=1...9), KOI8, WIN, ALT, SJIS, BIG5, WIN1250.

Функция требует PostgreSQL-7.0 или старше.

Ранее функция называлась pg_setclientencoding().

См. также: pg_client_encoding().

pg_client_encoding. Получение кодировки используемой клиентом

string **pg_client_encoding** ([int connection])

Возвращается одно из значений:

SQL_ASCII, EUC_JP, EUC_CN, EUC_KR, EUC_TW, UNICODE, MULE_INTERNAL, LATINX (X=1...9), KOI8, WIN, ALT, SJIS, BIG5, WIN1250.

Функция требует PostgreSQL-7.0 или старше.

Ранее функция называлась pg_clientencoding().

См. также: pg_set_client_encoding().

pg_trace. Разрешение трассировки подключения

bool **pg_trace** (string filename [, string mode [, int connection]])

Функция разрешает автоматическую запись обмена сообщениями между сервером PostgreSQL и его клиентской частью в файл отладки filename. Чтобы использовать эту возможность необходимо понимать внутренний протокол коммуникации PostgreSQL. В простейшем случае, причины ошибок, записанные в этом файле могут быть найдены с помощью команды: grep '^To backend' trace.log.

Аргументы filename и mode те же, что и в функции fopen() (mode по ум.: 'w'), connection указывает, какое подключение следует трассировать (если не указано, то используется последнее открытое).

Возвращает true если filename успешно открыт, или false при ошибке.

См. также: fopen() и pg_untrace().

pg_untrace. Запрет трассировки подключения

bool **pg_untrace** ([int connection])

Останавливает отладку, начатую функцией pg_trace(). Аргумент connection указывает, какое подключение следует трассировать (если не указано, то используется последнее открытое). Всегда возвращает TRUE.

См. также: pg_trace().

pg_exec. Выполнение запроса

int **pg_exec** (int connection, string query)

Посылает запрос query, для подключения connection.

Если при его выполнении запроса возникают ошибки, то функция возвращает FALSE. Получить комментарий ошибки можно функцией pg_errormessage().

Если запрос успешно выполнился, то возвращается набор записей, который может быть обработан функциями:

- pg_result() — получить элемент набора записей
- pg_fetch_array() — занести запись в массив
- pg_fetch_row() — занести запись в нумерованный массив
- pg_fetch_object() — занести запись в объект

Чтобы выяснить, сколько записей было возвращено командой SELECT, используйте функцию pg_num_rows(); а чтобы выяснить сколько записей было изменено в результате выполнения запросов DELETE, INSERT, REPLACE, или UPDATE, используйте функцию pg_cmdtuples().

После обработки результатов запроса, он может быть удален функцией pg_free_result(). Хотя в этом нет необходимости, так как ресурсы автоматически освобождаются при завершении скрипта.

pg_put_line. Пересылка серверу строку

bool **pg_put_line** ([resource connection_id, string data])

Для ускорения процедуры занесения информации в БД PostgreSQL позволяет метод непосредственной пересылки строк через поток ввода вывода. Возвращает true, или false при ошибке.

Конец строки (записи БД) маркируется символом "\n», а поля разделяются знаком табуляции "\t». Приложение должно явно послать пару символов "\." для указания, что клиент закончил отсылать данные.

```
<?php // скоростное добавление строк в таблицу
    $conn = pg_pconnect ("dbname=foo");
    pg_exec($conn, "create table tbl (a int4, b char(16), d float8)");
    pg_exec($conn, "copy tbl from stdin");  // начать запись
       pg_put_line($conn, "3\thello world\t4.5\n");
       pg_put_line($conn, "4\tgoodbye world\t7.11\n");
       pg_put_line($conn, "\\.\n");
    pg_end_copy($conn);
?>
```

См. также: pg_end_copy().

pg_end_copy. Синхронизация операции вставки

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

Функцию следует вызывать, после того как функцией pg_put_line() завершена передача данных серверу PostgreSQL (для того чтобы зафиксировать полученные данные). Возвращает true, или false при ошибке.

См. также: pg_put_line().

pg_numrows. Получение числа возвращенных записей

int **pg_numrows** (int result_id)

Возвращает число записей, возвращенных запросами SELECT. Дескриптор набора возвращенных записей (аргумент result_id) должен быть получен от функции pg_exec(). Возвращает.

pg_numfields. Получение числа полей в наборе записей

int **pg_numfields** (int result_id)

Возвращает число полей (столбцов) в наборе записей, возвращенном функцией pg_exec(); или -1 при ошибке.

pg_cmdtuples. Получение числа измененных записей в БД

int **pg_cmdtuples** (int result_id)

Возвращается число записей (сущностей), измененных в результате выполнения запросов DELETE, INSERT, REPLACE, или UPDATE. Если изменений произведено не было возвращает 0.

```
<?php
 $result = pg_exec ($conn, "INSERT INTO publisher VALUES ('Author')");
 $rows_affected = pg_cmdtuples ($result);
 echo "Число выполненных изменений: ", $rows_affected;
?>
```

pg_freeresult. Уничтожение набора записей

int **pg_freeresult** (int result_id)

Функция освобождает память, занимаемую набором записей result_id, возвращенным запросом. Ее следует использовать только, если требуется экономить память, так как память автоматически освобождается при завершении скрипта.

pg_result. Получение определенного элемента набора записей

mixed **pg_result** (int result_id, int row_number, mixed fieldname)

Функция возвращает значение из набора записей, возвращенного функцией pg_Exec(). Аргументы row_number и fieldname указывают соответственно

номер строки (записи) и поле (столбец). Нумерация начинается от 0. Поле можно указывать его именем или номером.

PostgreSQL имеет много встроенных типов данных и PHP поддерживаются только основные. Типы integer, boolean и oid возвращаются как целочисленные значения. Все формы float, и real типов возвращаются как дробные значения. Все остальные типы (включая массивы) возвращаются в виде строк в том же формате, что и в программе psql.

pg_fetch_array. Занесение записи в массив

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

Возвращает массив, соответствующий записи с номером `row`, из набора записей `result`, возвращенных запросом; или `false` если такой записи не имеется (также выдается предупреждение).

Данная функция является расширением функции `pg_fetch_row()`, и она может возвращать нумерованный или ассоциативный массив (или объединенный). Вид возвращаемого массива может указываться в аргументе `result_type` одной из констант: `PGSQL_NUM`, `PGSQL_ASSOC`, `PGSQL_BOTH` (по умолчанию).

В ассоциативных массивах индексами служат имена полей (они всегда в нижнем регистре).

Заметьте, что функция не значительно медленнее, чем `pg_fetch_row()`, но предоставляет дополнительную функциональность.

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

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

$arr = pg_fetch_array ($result, 0);
echo $arr[0] ." или ". $arr["author"] ;

// можно и так
for($i=0; $row = @pg_fetch_array($result,$i); $i++) {
      echo $row["author"];
}

?>
```

См. также: pg_fetch_row()

pg_fetch_row. Занесение записи в нумерованный массив

array **pg_fetch_row** (int result, int row)

Возвращает массив, соответствующий записи с номером row, из набора записей result, возвращенных запросом; или false если такой записи не имеется.

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

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

$num = pg_numrows($result);

for ($i=0; $i<$num; $i++) {
  $r = pg_fetch_row($result, $i);

  for ($j=0; $j<count($r); $j++) {
    echo "$r[$j] ";
  }
  echo "<BR>";
}
?>
```

См. также: pg_fetch_array(), pg_fetch_object(), pg_result().

pg_fetch_object. Получение записи в свойствах объекта

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

Возвращает объект, в свойствах которого находятся поля текущей записи; или false записей более не имеется.

По скорости функция идентична pg_fetch_array(), и почти идентична pg_fetch_row().

```
<?php
$database = "verlag";
$db_conn = pg_connect ("host=localhost port=5432 dbname=$database");
if (!$db_conn):
?><H1>Failed connecting to postgres database <?php echo $database ?></H1>
<?php
```

```
    exit;
endif;

$qu = pg_exec ($db_conn, "SELECT * FROM verlag ORDER BY autor");

for ($row = 0; $data = @pg_fetch_object ($qu, $row); $row++){
    echo $data->autor." (";
    echo $data->jahr ."): ";
    echo $data->titel."<BR>";
}
?><PRE><?php

$fields[] = Array ("autor", "Author");
$fields[] = Array ("jahr",  "  Year");
$fields[] = Array ("titel", " Title");

for ($row = 0; $data = @pg_fetch_object ($qu, $row); $row++) :
    reset ($fields);
    while (list (,$item) = each ($fields)):
        echo $item[1].": ".$data->$item[0]."\n";
    endwhile;
endfor;

echo "</PRE>";
pg_freeResult ($qu);
pg_close ($db_conn);
?>
```

См. также: `pg_fetch_array()` и `pg_fetch_row()`.

pg_fieldname. Определение имени поля

`string` **`pg_fieldname`** `(int result_id, int field_number)`

Возвращает имя поля по его номеру в наборе записей. Нумерация начинается с 0.

pg_fieldnum. Определение номера поля

`int` **`pg_fieldnum`** `(int result_id, string field_name)`

Возвращает номер поля в наборе записей по его имени; или –1 при ошибке.

pg_fieldtype. Определение типа поля

`string` **`pg_fieldtype`** `(int result_id, int field_number)`

pg_fieldprtlen. Длина значения в поле

`int` **`pg_fieldprtlen`** `(int result_id, int row_number, string field_name)`

Возвращает фактическую длину содержащегося в поле значения в байтах (число символов. Возвращает –1 при ошибке.

pg_fieldsize. Внутренний размер поля

`int` **`pg_fieldsize`** `(int result_id, int field_number)`

Возвращает размер в байтах, или значение –1 для полей переменной длины. При ошибке возвращается `false`.

pg_fieldisnull. Равно ли значение поля NULL?

`int` **`pg_fieldisnull`** `(int result_id, int row, mixed field)`

Возвращает 0, если поле `field` в записи `row` не содержит значение NULL; или 1, в противном случае. Поле может быть указано именем или номером.

pg_getlastoid. Получение идентификатора последнего объекта

`int` **`pg_getlastoid`** `(int result_id)`

Используется для получения oid назначенному вставленному объекту, последней SQL командой INSERT в функции `pg_exec()`. Возвращает положительное значение oid; или -1 если последняя команда в `pg_exec()` не была INSERT.

pg_locreate. Создание большого объекта

`int` **`pg_locreate`** `(int conn)`

Создает «Inversion Large Object» и возвращает его oid. `conn` содержит дескриптор подключения. Режимы доступа PostgreSQL: `INV_READ`, `INV_WRITE`, и `INV_ARCHIVE` не поддерживаются, объект всегда создается с доступом для чтения и записи. `INV_ARCHIVE` был удален из самого PostgreSQL (для версий 6.3 и старше).

pg_loopen. Открытие большого объекта

`int` **`pg_loopen`** `(int conn, int objoid, string mode)`

Открывает «Inversion Large Object» и возвращает его файловый дескриптор. Аргумент `objoid` указывает действительный oid (идентификатор объекта), а `mode` режим доступа, который может принимать значения: «r», «w», или «rw».

Не закрывайте подключение, не закрыв предварительно большой объект.

pg_loclose. Закрытие большого объекта

`void` **`pg_loclose`** `(int fd)`

В аргументе fd указывается дескриптор объекта, полученный от pg_loopen().

pg_loimport. Импорт большого объекта из файла

```
int pg_loimport (int file [, int connection_id])
```

Имя импортируемого файла (путь) указывается в аргументе filename. Возвращает идентификатор созданного объекта (oid), или FALSE при ошибке. Не забывайте, что работа с большими объектами должна проходить внутри транзакции.

pg_loexport. Экспорт большого объекта в файл

```
bool pg_loexport (int oid, int file [, int connection_id])
```

Имя файла (путь) указывается в аргументе filename, а идентификатор объекта в oid. Возвращает true, или false при ошибке. Не забывайте, что работа с большими объектами должна проходить внутри транзакции.

pg_loread. Чтение большого объекта

```
string pg_loread (int fd, int len)
```

Возвращает строку, прочитанную из объекта указанную его дескриптором fd. В аргументе len указывается максимальная длина возвращаемой строки.

pg_loreadall. Создание большого объекта и отсылка его содержимого непосредственно в поток вывода (браузеру)

```
void pg_loreadall (int fd)
```

Используется в основном для вывода двоичных данных (графики и звука).

pg_lowrite. Запись большого объекта

```
int pg_lowrite (int fd, string buf)
```

Записывает данные из буфера buf в объект, указанный дескриптором fd; и возвращает число фактически записанных байт, или false при ошибке.

pg_lounlink. Удаление большого объекта

```
void pg_lounlink (int conn, int lobjid)
```

pg_errormessage. Получение последнего сообщения об ошибке

```
string pg_errormessage (int connection)
```

Microsoft SQL Server

Ранее для доступа к Microsoft SQL 6.5 и 7.0 использовались разные модули, но они объединены в один.

mssql_connect. Подключение к серверу MS SQL

```
int mssql_connect ([string servername [, string username
   [, string password]]])
```

Возвращает дескриптор подключения к серверу MS SQL, или `false` при ошибке. Аргумент `servername` должен быть действительным именем, определенном в файле 'interfaces'.

Если функция повторно вызывается с теми же аргументами, новое подключение не создается, а возвращается идентификатор имеющегося.

Подключение к серверу закрывается при завершении скрипта или ранее, если явно вызывается функция `mssql_close()`.

См. также: `mssql_pconnect()`, `mssql_close()`.

mssql_close. Закрытие подключения к MS SQL Server

```
int mssql_close ([int link_identifier])
```

Возвращает `true`, или `false` при ошибке.

Идентификатор закрываемого подключения указывается в аргументе, если его не указывать, то закрывается последнее, открытое данным скриптом подключение.

Фактически использование данной функции не является обязательным, так как PHP автоматически закрывает все незакрытые неустойчивые подключения при завершении скрипта. Заметьте, устойчивые подключения, созданные функцией `mssql_pconnect()` не закрываются.

См. также: `mssql_connect()`, `mssql_pconnect()`.

mssql_pconnect. Создание устойчивого подключения к серверу MS SQL

```
int mssql_pconnect ([string servername [, string username [,
string password]]])
```

Возвращает дескриптор устойчивого подключения, или `false` при ошибке. `Mssql_pconnect()` действует подобно `Mssql_connect()` с двумя различиями:

Перед подключением, функция пытается проверить имеется ли уже открытое (устойчивое) подключение с параметрами (имя сервера, пользователя и пароль), аналогичными указанным. Если такое подключение обнаруживается, то возвращается его идентификатор, вместо создания нового подключения.

При завершении скрипта, подключение не закрывается, а остается действительным для дальнейшего использования. (Функция mssql_close() не может закрыть подключения, созданные с помощью mssql_pconnect()).

mssql_select_db. Выбор БД MS SQL

```
int mssql_select_db (string database_name
    [, int link_identifier])
```

Возвращает true, или false при ошибке.

Устанавливает БД с именем database_name активной для текущего подключения или указанного в link_identifier. Если подключений не имеется, то косвенно вызывается функция mssql_connect() с параметрами по умолчанию.

Последующие запросы, выполняемые функцией, будут адресованы данной БД.

См. также: mssql_connect(), mssql_pconnect(), и mssql_query()

mssql_query. Выполнение запроса к БД

```
int mssql_query (string query [, int link_identifier])
```

Посылает запрос текущей активной БД, для текущего (последнего открытого) подключения или указанного в link_identifier. Если подключений не имеется, то косвенно вызывается функция mssql_connect() с параметрами по умолчанию.

Возвращает дескриптор набора возвращенных записей, или false при ошибке.

См. также: mssql_select_db(), и mssql_connect().

mssql_num_rows. Получение числа возвращенных записей

```
int mssql_num_rows (string result)
```

Возвращает число записей в наборе result, возвращенном функцией mssql_query().

См. также: mssql_query(), и mssql_fetch_row().

В текущей реализации модуля отсутствует функция mssql_affected_rows() и для того, чтобы выяснить сколько записей было изменено командами UPDATE, INSERT или DELETE, приходится использовать код подобный следующему:

```
$rsRows = mssql_query("select @@rowcount as rows", $db);
$rows = mssql_result($rsRows, 0, "rows");
```

mssql_free_result. Уничтожение набора записей

```
int mssql_free_result (int result)
```

Функция освобождает память, занимаемую набором записей result, возвращенным запросом. Ее следует использовать только, если требуется экономить память, так как память автоматически освобождается при завершении скрипта.

mssql_result. Получение определенного элемента набора записей

int **mssql_result** (int result, int i, mixed field)

Возвращает содержимое ячейки из набора записей result. В аргументе row указывается номер записи (нумерация начинается с 0), в аргументе field можно указать индекс поля (число), имя поля или полное имя поля (вида: «имя_таблицы.имя_поля») или алиас поля (для запросов типа 'SELECT foo AS bar FROM...').

При работе с большими наборами записей более быстро выполняются функции типа _fetch_ (см. ниже). Также заметьте, что численные индексы обрабатываются быстрее строковых.

Не следует вызывать функцию mysql_result() в сочетании с другими функциями обработки набора записей.

Рекомендуется использовать альтернативные функции: mssql_fetch_row(), mssql_fetch_array(), и mssql_fetch_object().

mssql_data_seek. Перемещение внутреннего указателя записи

int **mssql_data_seek** (int result_identifier, int row_number)

При каждом вызове функции mssql_fetch_row() (или подобной), внутренний курсор записи смещается на следующую. Данная функция позволяет свободно перемещать курсор в наборе записей result_identifier, так, чтобы он указывал на запись с номером row_number (нумерация начинается с 0).

Она возвращает true, или false при ошибке.

См. также: mssql_data_seek().

mssql_fetch_array. Занесение записи в массив

array **mssql_fetch_array** (int result)

Возвращает массив, соответствующий текущей записи, из набора записей result, возвращенных запросом; или false записей более не имеется (при этом курсор записи перемещается на следующую).

Данная функция является расширением функции mysql_fetch_row(), и она возвращает объединенный нумерованно-ассоциативный массив. Доступ к элементам массива может осуществляться как через имена полей, так и по их номеру.

Заметьте, что функция не значительно медленнее, чем mysql_fetch_row(), но предоставляет дополнительную функциональность.

См. также: `mssql_fetch_row()`.

mssql_fetch_row. Занесение записи в нумерованный массив

`array` **`mssql_fetch_row`** `(int result)`

Возвращает массив, соответствующий текущей записи, из набора записей `result`, возвращенных запросом (последующий вызов функции возвращает следующую запись); или `false` записей более не имеется.

Каждое поле записи сохраняется в нумерованном элементе массива, (нумерация начинается с 0).

См. также: `mssql_fetch_array()`, `mssql_fetch_object()`, `mssql_data_seek()`, `mssql_fetch_length()`, и `mssql_result()`.

mssql_fetch_object. Получение записи в свойствах объекта

`int` **`mssql_fetch_object`** `(int result)`

Возвращает объект, в свойствах которого находятся поля текущей записи; или `false` записей более не имеется.

По скорости функция идентична `mysql_fetch_array()`, и почти идентична `mysql_fetch_row()`.

См. также: `mssql_fetch_array()` и `mssql_fetch_row()`.

mssql_num_fields. Получение числа полей в наборе записей

`int` **`mssql_num_fields`** `(int result)`

См. также: `mssql_query()`, `mssql_fetch_field()`, и `mssql_num_rows()`.

mssql_fetch_field. Получение информации о поле записи в свойствах объекта

`object` **`mssql_fetch_field`** `(int result [, int field_offset])`

Если номер поля `field_offset` не указан, при каждом вызове функции возвращаются свойства следующего поля из набора записей `result`.

Возвращаемый объект имеет следующие свойства (и содержит информацию):

- `name` – имя поля. Если поле - результат выполнения функции, имя возвращается в виде `computed#N`, где `#N` – порядковый номер.
- `column_source` — имя таблицы, которой принадлежит поле
- `max_length` — максимальная длинна поля
- `numeric` — 1, если поле числовое

См. также: `mssql_field_seek()`.

mssql_field_seek. Перемещение к указанному полю

```
int mssql_field_seek (int result, int field_offset)
```

Перемещается к указанному числом полю. И, если при следующем вызове `mssql_fetch_field()` не указывается номер поля, будет возвращено это.

См. также: `mssql_fetch_field()`.

InterBase

InterBase – популярная БД, производимая Borland/Inprise. http://www.interbase.com/. Последняя версия InterBase 6 распространяется бесплатно.

Эта БД использует одинарные кавычки (') для цитирования, по аналогии с БД Sybase, если в файл php.ini добавить опцию:

```
magic_quotes_sybase = On
```

ibase_connect. Подключение к серверу БД InterBase

```
int ibase_connect (string database [, string username [, string password [, string charset [, int buffers [, int dialect [, string role]]]]]])
```

Аргумент `database` должен указывать путь к файлу БД на сервере. Если сервер не локальный, то должен присутствовать префикс: 'hostname:' (TCP/IP), '//hostname/' (NetBEUI) или 'hostname@' (IPX/SPX), в зависимости от испоьзуемого протокола. `username` и `password` также могут быть указаны опциями конфигурации PHP `ibase.default_user` и `ibase.default_password`. `charset` определяет кодировку, используемую БД по умолчанию. `buffers` – число буферов используемых для кэширования БД (при значении 0 или не указании, сервер будет использовать собственное значение). `dialect` указывает используемый подключением диалект SQL (по ум., самый старший, поддерживаемый клиентской частью).

```
ibase_connect('localhost:/usr/db/db.gdb','SYSDBA','masterkey', 'ISO8859_1', '100', '1' ) ;
```

Если функция повторно вызывается с теми же аргументами, новое подключение не создается, а возвращается идентификатор имеющегося.

В конце скрипта принято закрывать подключение функцией `ibase_close()`, но это можно не делать, поскольку PHP автоматически закрывает все (неустойчивые) подключения при завершении скрипта.

```
<?php
    $host="localhost:c:\\IB6\\data\\db.gdb"; // WinNT путь
      $dbh = ibase_connect ($host, $username, $password);
    $stmt = 'SELECT * FROM tblname';
    $sth = ibase_query ($dbh, $stmt);
    while ($row = ibase_fetch_object ($sth)) {
```

```
        print $row->email . "\n";
    }
    ibase_close ($dbh);
?>
```

См. также: `ibase_pconnect()`.

ibase_close. Отключение от сервера БД InterBase

`int` **`ibase_close`** `([int connection_id])`

Возвращает `true`, или `false` при ошибке. Текущая транзакция (если имеется) завершается, а остальные отменяются.

Идентификатор закрываемого подключения указывается в аргументе, если его не указывать, то закрывается последнее, открытое данным скриптом подключение.

Фактически использование данной функции не является обязательным, так как PHP автоматически закрывает все незакрытые неустойчивые подключения при завершении скрипта.

Заметьте, устойчивые подключения, созданные функцией `mysql_pconnect()` не закрываются.

ibase_pconnect. Создание устойчивого подключения к серверу БД InterBase

`int` **`ibase_pconnect`** `(string database [, string username [, string password [, string charset [, int buffers [, int dialect [, string role]]]]]])`

`ibase_pconnect()` действует подобно `ibase_connect()` с двумя различиями:

Перед подключением, функция пытается проверить имеется ли уже открытое (устойчивое) подключение с параметрами, аналогичными указанным. Если такое подключение обнаруживается, то возвращается его идентификатор, вместо создания нового подключения. При завершении скрипта, подключение не закрывается, а остается действительным для дальнейшего использования. (Функция `ibase_close()` не может закрыть подключения, созданные с помощью `ibase_pconnect()`).

См. описание аргументов в: `ibase_connect()`.

ibase_prepare. Подготовка запроса для последующего выполнения

`int` **`ibase_prepare`** `([int link_identifier, string query])`

В дальнейшей части скрипта этот запрос можно выполнить функцией `ibase_execute()`, указав его параметры.

ibase_execute. Выполнение подготовленного запроса

`int` **`ibase_execute`** `(int query [, int bind_args ...])`

Выполняет запрос query, подготовленные функцией ibase_prepare(). Это более эффективно, чем использование для выполнения серии запросов функцией ibase_query(), если сами запросы различаются только указываемыми параметрами.

```
<?php
    $updates = array( 1 => 'Eric', 5 => 'Filip', 7 => 'Larry' );

    $query = ibase_prepare("UPDATE tbl SET A = ? WHERE Z = ?");

    while (list($a, $r) = each($updates)) {
        ibase_execute($query, $r, $a);
    }
?>
```

ibase_free_query. Уничтожение подготовленного запроса

int **ibase_free_query** (int query)

Уничтожает запрос, подготовленный функцией ibase_prepare().

ibase_query. Выполнение запроса к БД InterBase

int **ibase_query** ([int link_identifier, string query [, int bind_args]])

Возвращает дескриптор набора записей обрабатываемый: ibase_fetch_row(), ibase_fetch_object(), ibase_free_result() и ibase_free_query().

Хотя функция поддерживает возможность указания параметров запроса, целесообразнее использовать функции ibase_prepare() и ibase_execute().

ibase_trans. Начать транзакцию

int **ibase_trans** ([int trans_args [, int link_identifier]])

ibase_commit. Завершение транзакции

int **ibase_commit** ([int link_identifier, int trans_number])

Завершает транзакцию trans_number, созданную ibase_trans().

ibase_rollback. Отмена транзакции

int **ibase_rollback** ([int link_identifier, int trans_number])

Отменяет транзакцию trans_number, созданную ibase_trans().

ibase_free_result. Уничтожение набора записей

`int` **`ibase_free_result`** `(int result_identifier)`

Освобождает память, занятую набором записей, возвращенным `ibase_query()`.

ibase_fetch_row. Занесение записи в нумерованный массив

`array` **`ibase_fetch_row`** `(int result_identifier)`

Возвращает в массиве следующую запись из набора возвращенного `ibase_query()`.

ibase_fetch_object. Получение записи в свойствах объекта

`object` **`ibase_fetch_object`** `(int result_id)`

Возвращает в свойствах псевдо-объекта следующую запись из набора `result_id` возвращенного `ibase_query()` или `ibase_execute()`.

```
<php
    $dbh = ibase_connect ($host, $username, $password);
    $stmt = 'SELECT * FROM tblname';
    $sth = ibase_query ($dbh, $stmt);
    while ($row = ibase_fetch_object ($sth)) {
        print $row->email . "\n";
    }
    ibase_close ($dbh);
?>
```

См. также: `ibase_fetch_row()`.

ibase_field_info. Получение информации о поле записи в массиве

`array` **`ibase_field_info`** `(int result, int field number)`

Возвращает массив, содержащий элементы: `name`, `alias`, `relation`, `length`, `type`.

```
$rs=ibase_query("Select * from something");
$coln = ibase_num_fields($rs);
for ($i=0 ; $i < $coln ; $i++) {
    $col_info = ibase_field_info($rs, $i);
    echo "имя поля: ".$col_info['name']."\n";
    echo "синоним: ".$col_info['alias']."\n";
    echo "реляция: ".$col_info['relation']."\n";
    echo "размер: ".$col_info['length']."\n";
    echo "тип: ".$col_info['type']."\n";
    }
```

ibase_timefmt. Установка формата возвращаемых даты и времени

`int` **`ibase_timefmt`** `(string format [, int columntype])`

Внутренне поля даты и времени форматируются С-функцией `strftime()`, а с помощью данной функции можно установить вид форматирования возвращаемых запросами данных. В строке `format` дается шаблон, по которому производится форматирование. Аргумент `columntype` указывает, формат каких полей устанавливается (только с InterBase версии 6 или выше) и может принимать константные значения `IBASE_TIMESTAMP` (по умолчанию), `IBASE_DATE` и `IBASE_TIME`.

```
<?php
    // формат вида '04 hours 54 minutes'.
    ibase_timefmt("%H hours %M minutes", IBASE_TIME);
?>
```

Умолчания можно установить опциями конфигурации (php.ini): `ibase.timestampformat`, `ibase.dateformat` и `ibase.timeformat`.

ibase_num_fields. Число полей в наборе записей

`int` **`ibase_num_fields`** `(int result_id)`

В настоящее время функция еще не работоспособна.

```
<?php
    $dbh = ibase_connect ($host, $username, $password);
    $stmt = 'SELECT * FROM tblname';
    $sth = ibase_query ($dbh, $stmt);

    if (ibase_num_fields($sth) > 0) {
        while ($row = ibase_fetch_object ($sth)) {
            print $row->email . "\n";
        }
    } else {
        die ("No Results were found for your query");
    }

    ibase_close ($dbh);
?>
```

См. также: `ibase_field_info()`.

ibase_errmsg. Получение сообщения об ошибке

`string` **`ibase_errmsg`** `(void)`

Поля BLOB

В PHP имеются следующие функции для работы с полями BLOB БД InterBase (документация для них пока отсутствует):

`ibase_blob_create` – создает поле BLOB, которое затем может быть сохранено запросом (возвращенный дескриптор передается как параметр запроса)

`ibase_blob_open` – открывает возвращенное запросом поле BLOB для операций

`ibase_blob_close` – закрывает поле BLOB

`ibase_blob_add` – добавляет в конец поля BLOB данные

`ibase_blob_import` – записывает содержимое открытого файла в поле BLOB

`ibase_blob_get` – читает одержимое поля BLOB

`ibase_blob_echo` – выводит содержимое поля BLOB

`ibase_blob_info`

`ibase_blob_cancel`

Поскольку единственной информацией об этих функциях является пример из исходников, он приводится далее.

```
<?
$cnn=ibase_connect("localhost:F:\interbase\data\aaa\iq.gdb",
            "sysdba","masterkey") or die ("Can't connect");
ibase_query("create table tbl (v_integer integer, v_blob blob)");
ibase_commit();

$name = "F:\\php_src\\ext\\interbase\\blob.tmp";
$f = fopen($name,"r");
$bl_s = ibase_blob_import($f);
ibase_query("insert into tbl (v_integer, v_blob) values (1, ?)", $bl_s);

$bl_h = ibase_blob_create();
$fp = fopen($name,"r");
while($piece = fread($fp, 1024)){
        ibase_blob_add($bl_h, $piece);
}
fclose($ftmp);
ibase_blob_add($bl_h, "+---------------------------------+\n");
$bl_s = ibase_blob_close($bl_h);
ibase_query("insert into tbl (v_integer, v_blob) values (2, ?)", $bl_s);

$q = ibase_query("select v_blob from tbl where v_integer = 2");
$row = ibase_fetch_object($q,IBASE_TEXT);
echo $row->V_BLOB;

$q = ibase_query("select v_blob from tbl where v_integer = 2");
```

```
$row = ibase_fetch_object($q);
ibase_blob_echo($row->V_BLOB);

$q = ibase_query("select v_blob from tbl where v_integer = 2");
$row = ibase_fetch_object($q);
$bl_h = ibase_blob_open($row->V_BLOB);
while($piece = ibase_blob_get($bl_h, 1024))
     $blob .= $piece;
ibase_blob_close($bl_h);
ibase_free_result($q);

ibase_close();
?>
```

Informix

Существующий модуль расширения Informix позволяет работать с БД Informix (IDS) 7.x, SE 7.x, Universal Server (IUS) 9.x и IDS 2000. Поддержка IDS 7.x полная (включая поля BYTE и TEXT), но для IUS 9.x еще не завершена поддержка типов SLOB и CLOB.

Для компиляции необходима ESQL/C (версия 7.2x или выше должна входить в Informix Client SDK) для компиляции драйвера Informix. Также должны быть установлены значения переменных системы INFORMIXDIR и добавлена директория $INFORMIXDIR/bin в список путей PATH до запуска скрипта configure (с ключом --with_informix=yes). Автоопределение директорий файлов заголовков и библиотек может быть переопределено установкой системных переменных IFX_LIBDIR, IFX_LIBS и IFX_INCDIR.

Во время выполнения переменные системы `INFORMIXDIR`, `INFORMIXSERVER` и `PATH` должны быть правильно инициализированы.

Для полей BLOB (TEXT и BYTE) запросом возвращаются их идентификаторы. Их содержимое можно получить в строковой переменной (если установлено `ifx_blobinfile(0);`) функцией `ifx_get_blob($blob_id);` или сохранить в файле (если установлено `ifx_blobinfile(1);`) функцией `ifx_get_blob($blob_id);`.

ifx_connect. Подключение к серверу БД Informix

`int` **`ifx_connect`** `([string database [, string user [, string password]]])`

Все аргументы необязательны, и если их не указывать, то используются значения из файла конфигурации: `ifx.default_host` (если не указано, библиотеки Informix используют значение системной переменной `INFORMIXSERVER`), `ifx.default_user`, `ifx.default_password`. Возвращается дескриптор подключения или `FALSE` при ошибке.

```
$conn_id = ifx_connect ("mydb@ol_srv1", "imyself", "mypassword");
```

Если функция повторно вызывается с теми же аргументами, новое подключение не создается, а возвращается идентификатор имеющегося.

В конце скрипта принято закрывать подключение функцией `ifx_close()`, но это можно не делать, поскольку PHP автоматически закрывает все (неустойчивые) подключения при завершении скрипта.

См. также: `ifx_pconnect()`, и `ifx_close()`.

ifx_pconnect. Создание устойчивого подключения к серверу БД Informix

```
int ifx_pconnect ([string database [, string userid [, string password]]])
```

Возвращается дескриптор подключения или `FALSE` при ошибке.

`ifx_pconnect()` действует подобно `ifx_connect()` с двумя различиями:

Перед подключением, функция пытается проверить имеется ли уже открытое (устойчивое) подключение с параметрами (БД, пользователь, пароль), аналогичными указанным. Если такое подключение обнаруживается, то возвращается его идентификатор, вместо создания нового подключения. При завершении скрипта, подключение не закрывается, а остается действительным для дальнейшего использования. (Функция `ifx_close()` не может закрыть подключения, созданные с помощью `ifx_pconnect()`).

См. также: `ifx_connect()`.

ifx_close. Закрытие подключения к Informix

```
int ifx_close ([int link_identifier])
```

Всегда возвращает `true`.

Идентификатор закрываемого подключения указывается в аргументе, если его не указывать, то закрывается последнее, открытое данным скриптом подключение.

Фактически использование данной функции не является обязательным, так как PHP автоматически закрывает все незакрытые неустойчивые подключения при завершении скрипта.

Устойчивые подключения, созданные функцией `ifx_pconnect()` не закрываются.

```
$conn_id = ifx_connect ("mydb@ol_srv", "itsme", "mypassword");
...
ifx_close($conn_id);
```

См. также: `ifx_connect()`, и `ifx_pconnect()`.

ifx_query. Выполнение запроса к БД Informix

`int` **`ifx_query`** `(string query [, int link_identifier [, int cursor_type [, mixed blobidarray]]])`

Посылает запрос текущей активной БД, для текущего (последнего открытого) подключения или указанного в `link_identifier`. Если подключений не имеется, то косвенно вызывается функция `ifx_connect()` с параметрами по умолчанию.

Возвращает дескриптор набора возвращенных записей, или `false` при ошибке.

Для запросов «SELECT» необязательным аргументом `cursor_type` можно указать специальный тип курсора (значение формируется битовой маской из констант `IFX_SCROLL`, `IFX_HOLD`).

В зависимости от типа запроса число возвращенных (измененных) записей (приблизительное или фактическое) можно выяснить функцией `ifx_affected_rows()`.

Если в запросе передаются поля BLOB (BYTE или TEXT), то в аргументе (массиве) `blobidarray` указываются дескрипторы (blob ids), а в запросе вместо их значений ставится знак "?".

Если содержимое полей TEXT (или BYTE) допускает, можно использовать функции `ifx_textasvarchar(1)` и `ifx_byteasvarchar(1)`, что позволяет работать с этими полями (получать их содержимое), как если бы они были обычными (но длинными) полями VARCHAR, не заботясь о blob дескрипторах. В противном случае придется использовать дополнительные функции.

В ситуации `ifx_textasvarchar(0)` и `ifx_byteasvarchar(0)` (по умолчанию), запросы выборки возвращают для полей BLOB их дескрипторы (целые числа). И содержимое можно получить специальными функциями.

```
// Пример 1. распечатать всю таблицу "orders" в виде html
ifx_textasvarchar(1);      // use "text mode" for blobs
$res_id = ifx_query("select * from orders", $conn_id);
if (! $res_id) {
    printf("Can't select orders : %s\n<br>%s<br>\n", ifx_error());
    ifx_errormsg();
    die;
}
ifx_htmltbl_result($res_id, "border=\"1\"");
ifx_free_result($res_id);


// Пример 2. добавление записей в таблицу "catalog"
                     // создать blob поля
$textid = ifx_create_blob(0, 0, "Text column in memory");
$byteid = ifx_create_blob(1, 0, "Byte column in memory");
                     // внести blob id в массив
$blobidarray[] = $textid;
$blobidarray[] = $byteid;
```

```
                        // выполнить запрос
$query = "insert into catalog (stock_num, manu_code, " .
         "cat_descr,cat_picture) values(1,'HRO',?,?)";
$res_id = ifx_query($query, $conn_id, $blobidarray);
if (! $res_id) {   ... error ...   }

ifx_free_result($res_id);
```

См. также: `ifx_connect()`.

ifx_prepare. Подготовка SQL-запроса для последующего выполнения

`int` **`ifx_prepare`** `(string query, int conn_id [, int cursor_def, mixed blobidarray])`

Возвращает дескриптор, используемый в функции `ifx_do()`. Для запросов «SELECT» необязательным аргументом `cursor_type` можно указать тип курсора (значение формируется битовой маской из констант `IFX_SCROLL`, `IFX_HOLD`).

Если в запросе передаются поля BLOB (BYTE или TEXT), то в аргументе (массиве) `blobidarray` указываются дескрипторы (blob ids), а в запросе вместо их значений ставится знак "?".

Если содержимое полей TEXT (или BYTE) допускает, можно использовать функции `ifx_textasvarchar(1)` и `ifx_byteasvarchar(1)`, что позволяет работать с этими полями (получать их содержимое), как если бы они были обычными (но длинными) полями VARCHAR, не заботясь о blob дескрипторах. В противном случае придется использовать дополнительные функции.

Подготовленные запросы позволяют сперва вычислить число подходящих запросу записей функцией `ifx_affected_rows()`.

См. также: `ifx_do()`.

ifx_do. Выполнение подготовленного запроса

`int` **`ifx_do`** `(int result_id)`

Возвращает `true`, или `false` при ошибке.

См. также: `ifx_prepare()`, `ifx_affected_rows()`.

ifx_error. Получение кода ошибки последней операции Informix

`string` **`ifx_error`** `(void);`

Возвращаемая строка описания ошибки форматируется следующим образом:

x [`SQLSTATE` = (две цифры) (три цифры) `SQLCODE`=cccc]

Выражение x — один из символов, приведенных в следующем спсике.

- пробел — без ошибки.

- E — ошибка.
- N — больше нет данных.
- W — предупреждение.
- ? — неопределенное сообщение.

Если x-символ не пробел, то SQLSTATE и SQLCODE описывают детали ошибки (см. документацию БД).

См. также: `ifx_errormsg()`

ifx_errormsg. Получение описания ошибки последней операции

`string` **`ifx_errormsg`** `([int errorcode])`

Если в аргументе указать код ошибки, то будет получено ее описание.

```
printf("%s\n<br>", ifx_errormsg(-201));
```

См. также: `ifx_error()`

ifx_affected_rows. Получение числа измененных или возвращенных записей

`int` **`ifx_affected_rows`** `(int result_id)`

В аргументе указывается дескриптор набора записей, возвращенный функцией `ifx_query()` или `ifx_prepare()`.

Для запросов типа insert, update и delete возвращается число фактически добавленных, измененных, удаленных записей (значение `sqlerrd[2]`). Для запросов выборки select, это предполагаемое число (`sqlerrd[0]`), которое не всегда верно.

```
$rid = ifx_prepare ("SELECT * FROM emp
                     WHERE name LIKE " . $name, $connid);
if (! $rid) {
    ... error ...
}
$rowcount = ifx_affected_rows ($rid);
if ($rowcount > 1000) {
    printf ("Критерию соответствует слишком много записей (%d)\n",
             $rowcount);
    die ("Ограничте условия запроса <br>\n");
}
```

См. также: `ifx_num_rows()`

ifx_getsqlca. Получение параметров запроса sqlca.sqlerrd[0..5]

`array` **`ifx_getsqlca`** `(int result_id)`

В аргументе указывается дескриптор набора записей, возвращенный функцией ifx_query() или ifx_prepare(). Возвращается ассоциативный массив, содержащий дополнительные результаты выполнения запроса сервером sqlerrd0 ... sqlerrd5. Альтернативно ту же операцию можно выполнить запросом «SELECT dbinfo('sqlca.sqlerrdx')".

```
// предполагается что первое поле таблицы tbl последовательное
$qid = ifx_query("INSERT INTO tbl
          VALUES (0, '2nd column', 'another column') ", $connid);
if (! $qid) {     ... error ...   }
$sqlca = ifx_getsqlca ($qid);
echo "Последовательное значение добавленной записи: " .
         $sqlca["sqlerrd1"];
```

ifx_fetch_row. Занесение записи в массив

array **ifx_fetch_row** (int result_id [, mixed position])

Возвращает в ассоциативном массиве (индексами служат имена полей) запись из результата запроса (возвращенного функциями ifx_query() или ifx_prepare()), или false если записей больше не имеется.

Для полей Blob возвращаются их цифровые идентификаторы, которые можно использовать в функции ifx_get_blob() (если конечно, не использовались ifx_textasvarchar(1) или ifx_byteasvarchar(1)).

Необязательным аргументом position для SCROLL курсоров можно указать какую запись следует возвратить: «NEXT», «PREVIOUS», «CURRENT», «FIRST», «LAST» или «абсолютный» номер записи.

Последующие вызовы (по умолчанию возвращают следующие записи).

```
$rid = ifx_prepare ("SELECT * FROM emp WHERE name LIKE " . $name,
                    $connid, IFX_SCROLL);
if (! $rid) {   ... error ...   }
$rowcount = ifx_affected_rows($rid);
if ($rowcount > 1000) {
    printf ("слишком много записей (%d)\n<br>", $rowcount);
    die ("Ограничте критерий <br>\n");
}
if (! ifx_do ($rid)) {   ... error ... }
$row = ifx_fetch_row ($rid, "NEXT");
while (is_array($row)) {
    for(reset($row); $fieldname=key($row); next($row)) {
        $fieldvalue = $row[$fieldname];
        printf ("%s = %s,", $fieldname, $fieldvalue);
    }
    print("\n<br>");
    $row = ifx_fetch_row ($rid, "NEXT");
}
```

```
ifx_free_result ($rid);
```

ifx_htmltbl_result. Вывод результатов запроса в таблице HTML

```
int ifx_htmltbl_result (int result_id [, string html_table_options])
```

Возвращает число выведенных записей или FALSE при ошибке.

Во втором аргументе можно указать атрибуты тега <table>.

```
$rid = ifx_prepare ("select * from emp where name like " . $name,
                    $connid, IFX_SCROLL);
if (! $rid) {   ... error ...  }
$rowcount = ifx_affected_rows ($rid);
if (! ifx_do($rid) {  ... error ... }

ifx_htmltbl_result ($rid, "border=\"2\"");

ifx_free_result($rid);
```

ifx_fieldtypes. Получение списка полей набора записей

```
array ifx_fieldtypes (int result_id)
```

Возвращает ассоциативный массив (имена полей являются индексами, а SQL типы полей – значениями); или FALSE при ошибке. В аргументе указывается результат запроса.

```
$types = ifx_fieldtypes ($resultid);
if (! isset ($types)) {  ... error ...   }
for ($i = 0; $i < count($types); $i++) {
    $fname = key($types);
    printf("%s :\t type =  %s\n", $fname, $types[$fname]);
    next($types);
}
```

ifx_fieldproperties. Получение списка свойств полей

```
array ifx_fieldproperties (int result_id)
```

Возвращает ассоциативный массив (имена полей являются индексами, а SQL свойства полей – значениями); или FALSE при ошибке. В аргументе указывается результат запроса.

Свойства кодируются в виде: «SQLTYPE;length;precision;scale;ISNULLABLE», где SQLTYPE – тип поля Informix, (например «SQLVCHAR»), а ISNULLABLE = «Y» или «N».

```
$properties = ifx_fieldproperties ($resultid);
```

```
if (! isset($properties)) {  ... error ... }
for ($i = 0; $i < count($properties); $i++) {
    $fname = key ($properties);
    printf ("%s:\t type =  %s\n", $fname, $properties[$fname]);
    next ($properties);
}
```

ifx_num_fields. Получение числа полей возвращаемых запросом

```
int ifx_num_fields (int result_id)
```

ifx_num_rows. Получение числа записей уже полученных из результата запроса

```
int ifx_num_rows (int result_id)
```

ifx_free_result. Уничтожение результата запроса

```
int ifx_free_result (int result_id)
```

ifx_create_char. Создание char объекта

```
int ifx_create_char (string param)
```

В аргументе указывается содержимое объекта.

ifx_free_char. Уничтожение char объекта

```
int ifx_free_char (int bid)
```

Возвращает `true`, или `false` при ошибке.

ifx_update_char. Изменение содержимого char объекта

```
int ifx_update_char (int bid, string content)
```

Возвращает `true`, или `false` при ошибке.

ifx_get_char. Получение содержимого char объекта

```
int ifx_get_char (int bid)
```

ifx_create_blob. Создание blob объекта

```
int ifx_create_blob (int type, int mode, string param)
```

Аргумент `type` указывает тип: 1 = TEXT, 0 = BYTE;

`mode`: 0 – что содержимое объекта будет сохраняться в памяти, 1 – в файле;

param: если mode = 0, это содержимое объекта, а если mode = 1, то это дескриптор файла данных.

Возвращает идентификатор созданного объекта, или false при ошибке.

ifx_copy_blob. Клонирование blob объекта

```
int ifx_copy_blob (int bid)
```

Возвращает идентификатор созданного объекта, или false при ошибке.

ifx_free_blob. Закрытие blob объекта

```
int ifx_free_blob (int bid)
```

Возвращает true, или false при ошибке.

ifx_get_blob. Получение содержимого blob объекта

```
int ifx_get_blob (int bid)
```

В аргументе указывается идентификатор, полученный запросом.

Если установлено ifx_blobinfile(1), возвращается имя созданного файла, в котором сохранено содержимое поля. Ответственность за удаление этого временного файла возлагается на вас. Размещение его зависит от переменной окружения «blobdir», (по умолчанию = ".", то есть в текущей директории); файлы имеют префикс «blb». Для облегчения очистки используйте, например: putenv(«blobdir=tmpblob»);.

ifx_update_blob. Изменение содержимого объекта blob

```
int ifx_update_blob (int bid, string content)
```

Возвращает true, или false при ошибке.

ifx_blobinfile_mode. Установка режима получения содержимого объектов blob

```
void ifx_blobinfile_mode (int mode)
```

Если указывается значение «0» то содержимое будет сохраняться в памяти, а если «1», то в файле. Умолчание также устанавливается в файле конфигурации.

ifx_textasvarchar. Установка режима получения полей TEXT

```
void ifx_textasvarchar (int mode)
```

Если указывается значение «0» то для полей TEXT запросом будет возвращаться их идентификатор, а если «1», то непосредственно их содержимое. Умолчание также устанавливается в файле конфигурации.

ifx_byteasvarchar. Установка режима получения полей BYTE

```
void ifx_byteasvarchar (int mode)
```

Если указывается значение «0» то для полей BYTE запросом будет возвращаться их идентификатор, а если «1», то непосредственно их содержимое. Умолчание также устанавливается в файле конфигурации.

ifx_nullformat. Установка формат значений NULL

```
void ifx_nullformat (int mode)
```

Устанавливает, какое значение будет возвращаться запросом для значений NULL. Если указывается значение `mode` «0» будет возвращаться пустая строка "", а если «1», то значение «NULL». Умолчание также устанавливается в файле конфигурации.

ifxus_create_slob. Создание объекта slob и его открытие

```
int ifxus_create_slob (int mode)
```

Аргументом можно указать режим открытия (возможны битовые комбинации): 1 = LO_RDONLY, 2 = LO_WRONLY, 4 = LO_APPEND, 8 = LO_RDWR, 16 = LO_BUFFER, 32 = LO_NOBUFFER. Также возможно использовать именованные константы IFX_LO_RDONLY, IFX_LO_WRONLY и т. п. Возвращает идентификатор созданного объекта, или `false` при ошибке.

ifxus_free_slob. Удаление объекта slob

```
int ifxus_free_slob (int bid)
```

Возвращает `true`, или `false` при ошибке.

ifxus_close_slob. Закрытие slob объекта

```
int ifxus_close_slob (int bid)
```

Возвращает `true`, или `false` при ошибке.

ifxus_open_slob. Открытие объекта slob

```
int ifxus_open_slob (long bid, int mode)
```

Аргументом `mode` можно указать режим открытия (возможны битовые комбинации): 1 = LO_RDONLY, 2 = LO_WRONLY, 4 = LO_APPEND, 8 = LO_RDWR, 16 = LO_BUFFER, 32 = LO_NOBUFFER. Также возможно использовать именованные константы IFX_LO_RDONLY, IFX_LO_WRONLY и т. п. Возвращает идентификатор созданного объекта, или `false` при ошибке.

ifxus_tell_slob. Получение позицию курсора

```
int ifxus_tell_slob (long bid)
```

Возвращает текущую позицию (чтения/записи) открытого slob объекта `bid`; или `FALSE` при ошибке.

ifxus_seek_slob. Установка позиции курсора

```
int ifxus_seek_blob (long bid, int mode, long offset)
```

Устанавливает позицию (чтения/записи) `offset` в байтах, открытого slob объекта `bid`. Возвращает позицию или `FALSE` при ошибке. В аргументе `mode` указывается, от какого места отчитывается смещение: 0 = LO_SEEK_SET (от начала), 1 = LO_SEEK_CUR (от текущей позиции), 2 = LO_SEEK_END (от конца).

ifxus_read_slob. Чтение данных из объекта slob

```
string ifxus_read_slob (long bid, long nbytes)
```

Возвращает прочитанную строку данных размером `nbytes` байт из slob объекта `bid` или `FALSE` при ошибке.

ifxus_write_slob. Запись данных в объект slob

```
int ifxus_write_slob (long bid, string content)
```

Записывает строку данных `content` в slob объект `bid`. Возвращает число записанных байт или `false` при ошибке.

Ingres II

Для использования этих функций, необходимо скомпилировать PHP с поддержкой Ingres используя опцию --with-ingres. Для этого необходима библиотека Open API и заголовочные файлы включенные в Ingres II. Если переменная окружения II_SYSTEM не установлена корректно, можно использовать опцию --with-ingres=DIR, указывая директорию размещения Ingres.

Если при использовании расширения Apache, веб сервер не запускается и сообщает: «PHP Fatal error: Unable to start ingres_ii module in Unknown on line 0», удостоверьтесь, что переменная окружения II_SYSTEM установлена корректно. Добавление «`export II_SYSTEM="/home/ingres/II`» в скрипт, запускающий Apache, перед строкой запуска httpd должно помочь.

Ingres не допускает конкурентные запросы/транзакции для одного подключения. Поэтому результат запроса должен обрабатываться до выполнения следующего запроса, а транзакция совершаться (или отменяться) до начала следующей транзакции.

ingres_connect. Подключение к серверу Ingres II

```
resource ingres_connect ([string database [, string username [,
string password]]])
```

Возвращает дескриптор подключения Ingres II, или `false` при ошибке.

Аргумент `database` имеет синтаксис: `[node_id::]dbname[/svr_class]`.

Для неуказанных параметров, используются значения из файла конфигурации php.ini `ingres.default_database`, `ingres.default_user` и `ingres.default_password`.

Поскольку все ingres функции используют дескриптор последнего открытого подключения по умолчанию, его не требуется сохранять, ее имеется всего одно подключение. Подключение закрывается при завершении скрипта или явном вызове `ingres_close()`.

```
<?php
    $link = ingres_connect ("mydb", "user", "pass")
        or die ("Could not connect");
    print ("Connected successfully");
    ingres_close ($link); // или просто ingres_close();
?>
```

См. также: `ingres_pconnect()`, и `ingres_close()`.

ingres_pconnect. Создание устойчивого подключения к БД сервера Ingres II

`resource` **`ingres_pconnect`** `([string database [, string username [, string password]]])`

Возвращает дескриптор подключения, или `false` при ошибке.

`ingres_pconnect()` действует подобно `ingres_connect()` с двумя различиями:

Перед подключением, функция пытается проверить имеется ли уже открытое (устойчивое) подключение с параметрами (имя БД, пользователя и пароль), аналогичными указанным. Если такое подключение обнаруживается, то возвращается его идентификатор, вместо создания нового подключения.

При завершении скрипта, подключение не закрывается, а остается действительным для дальнейшего использования. (Функция `ingres_close()` не может закрыть подключения, созданные с помощью `ingres_pconnect()`).

См. также: `ingres_connect()`, и `ingres_close()`.

ingres_close. Закрытие подключения

`bool` **`ingres_close`** `([resource link])`

Возвращает `true`, или `false` при ошибке.

Идентификатор закрываемого подключения указывается в аргументе, если его не указывать, то закрывается последнее, открытое данным скриптом подключение.

Фактически использование данной функции не является обязательным, так как PHP автоматически закрывает все незакрытые неустойчивые подключения при завершении скрипта. Заметьте, устойчивые подключения, не закрываются.

См. также: `ingres_connect()`, и `ingres_pconnect()`.

ingres_query. Выполнение запроса к БД

`bool` **`ingres_query`** `(string query [, resource link])`

Возвращает `true`, или `false` при ошибке.

Запрос становится частью текущей транзакции (при необходимости открывается новая). Для завершения транзакции вызывается `ingres_commit()`, а с помощью `ingres_rollback()` можно отменить транзакцию. Функцией `ingres_autocommit()` можно установить режим немедленного выполнения запроса. Некоторые запросы SQL не могут выполняться данной функцией: «get dbevent», «prepare to commit», «savepoint», и все запросы использующие курсор.

```
<?php
ingres_connect ($database, $user, $password);

ingres_query ("select * from table");
while ($row = ingres_fetch_row()) {
    echo $row[1];
    echo $row[2];
}
?>
```

См. также: `ingres_fetch_array()`, `ingres_fetch_object()`, `ingres_fetch_row()`, `ingres_commit()`, `ingres_rollback()` и `ingres_autocommit()`.

ingres_rollback. Отмена транзакции

`bool` **`ingres_rollback`** `([resource link])`

Закрывает текущую транзакцию, отменяя все изменения внесенные запросами от начала транзакции. Новая транзакция открывается при последующем вызове `ingres_query()`.

См. также: `ingres_query()`, `ingres_commit()` и `ingres_autocommit()`.

ingres_commit. Завершение транзакции

`bool` **`ingres_commit`** `([resource link])`

Закрывает текущую транзакцию, фиксируя все изменения внесенные запросами от начала транзакции. Новая транзакция открывается при последующем вызове `ingres_query()`. Функцией `ingres_autocommit()` можно установить режим немедленного выполнения каждого запроса.

См. также: ingres_query(), ingres_rollback() и ingres_autocommit().

ingres_autocommit. Переключение режима немедленного выполнения запроса

bool **ingres_autocommit** ([resource link])

При запуске скрипта, по умолчанию, запрос(-ы) (выполненный функцией ingres_query()) необходимо завершать явно, вызовом функции ingres_commit() (или отменять транзакцию с помощью ingres_rollback()).

Если же (вызовом данной функции) переключить режим автовыполнения, то запросы будут выполняться немедленно, то есть после каждого запроса косвенно вызывается ingres_commit().

Повторный вызов функции переключает режим на противоположный.

См. также: ingres_query(), ingres_rollback() и ingres_commit().

ingres_num_rows. Число измененных или возвращенных запросом записей

int **ingres_num_rows** ([resource link])

После запросов delete, insert, update функция возвращает число измененных записей, а после запросов выборки – число возвращенных записей (функция должна вызываться до разборки результата запроса функциями: ingres_fetch_array(), ingres_fetch_object() или ingres_fetch_row()).

В цикле обработки результатов запроса целесообразнее использовать возвращаемое fetch-функциями значение, для проверки того, все ли записи были обработаны.

См. также: ingres_query(), ingres_fetch_array(), ingres_fetch_object() и ingres_fetch_row().

ingres_num_fields. Число возвращенных запросом полей

int **ingres_num_fields** ([resource link])

См. также: ingres_query(), ingres_fetch_array(), ingres_fetch_object() и ingres_fetch_row().

ingres_fetch_row. Занесение записи в нумерованный массив

array **ingres_fetch_row** ([resource link])

Возвращает в массиве (нумерация полей начинается с 1) следующую запись возвращенную запросом; или false если все записи уже были обработаны.

```
<?php
ingres_connect ($database, $user, $password);

ingres_query ("select * from table");
while ($row = ingres_fetch_row()) {
    echo $row[1];  // первое поле записи
```

```
    echo $row[2];
}
?>
```

См. также: ingres_num_fields(), ingres_query(), ingres_fetch_array() и ingres_fetch_object().

ingres_fetch_array. Занесение записи в массив

array **ingres_fetch_array** ([int result_type [, resource link]])

Возвращает в массиве следующую запись, возвращенную запросом; или false если записей больше не имеется.

Данная функция является расширением функции ingres_fetch_row(), и она может возвращать нумерованный или ассоциативный массив (или объединенный). Вид возвращаемого массива может указываться в аргументе result_type одной из констант: II_NUM, II_ASSOC, II_BOTH (по умолчанию). В ассоциативных массивах индексами служат имена полей.

Если имеются одноименные поля, используется последнее имя. Для доступа к одноименным полям можно использовать числовые индексы (поля нумеруются в той последовательности, в которой указаны в запросе или в таблице) или алиасы.

```
ingres_query(select t1.f1 as f t2.f1 as b from t1, t2);
$result = ingres_fetch_array();
$foo = $result["f"];
$bar = $result["b"];
```

Заметьте, что функция не значительно медленнее, чем ingres_fetch_object(), но предоставляет дополнительную функциональность.

```
<?php
ingres_connect ($database, $user, $password);

ingres_query ("select * from table");
while ($row = ingres_fetch_array()) {
    echo $row["user_id"];  # используем ассоциативный массив
    echo $row["fullname"];
    echo $row[1];          # используем нумерованный массив
    echo $row[2];
}
?>
```

См. также: ingres_query(), ingres_num_fields(), ingres_field_name(), ingres_fetch_object() и ingres_fetch_row().

ingres_fetch_object. Получение записи в свойствах объекта

`object` **`ingres_fetch_object`** `([int result_type [, resource link]])`

Возвращает объект, в свойствах которого находятся поля текущей записи; или `false` записей более не имеется. Заметьте осуществлять доступ к полям по их номерам невозможно (объект не может иметь численные свойства).

По скорости функция идентична `ingres_fetch_array()`, и почти идентична `ingres_fetch_row()`.

```
<?php
ingres_connect ($database, $user, $password);
ingres_query ("select * from table");
while ($row = ingres_fetch_object()) {
    echo $row->user_id;
    echo $row->fullname;
}
?>
```

См. также: `ingres_query()`, `ingres_num_fields()`, `ingres_field_name()`, `ingres_fetch_array()` и `ingres_fetch_row()`.

ingres_field_name. Имя поля записи в результате запроса

`string` **`ingres_field_name`** `(int index [, resource link])`

Функция возвращает имя поля с индексом `index` (нумерация начинается с 1) набора записей возвращенных запросом; или `false` при ошибке.

См. также: `ingres_query()`, `ingres_num_fields()`, `ingres_fetch_array()`, `ingres_fetch_object()` и `ingres_fetch_row()`.

ingres_field_type. Получение типа поля

`string` **`ingres_field_type`** `(int index [, resource link])`

Функция возвращает имя поля с индексом `index` (нумерация начинается с 1) набора записей возвращенных запросом; или `false` при ошибке.

Возвращаемые типы полей: `IIAPI_BYTE_TYPE`, `IIAPI_CHA_TYPE`, `IIAPI_DTE_TYPE`, `IIAPI_FLT_TYPE`, `IIAPI_INT_TYPE`, `IIAPI_VCH_TYPE`. Некоторые типы могут соответствовать нескольким SQL типам, в зависимости от длинны поля (см. `ingres_field_length()`). Например, `IIAPI_FLT_TYPE` может быть либо float4 или float8. См. документацию: «Ingres/OpenAPI User Guide - Appendix C».

См. также: `ingres_query()`, `ingres_num_fields()`, `ingres_fetch_array()`, `ingres_fetch_object()` и `ingres_fetch_row()`.

ingres_field_nullable. Может ли поле принимать значение null?

`bool` **`ingres_field_nullable`** `(int index [, resource link])`

Возвращает `true` если поле с индексом `index` (нумерация начинается с 1) может иметь значение NULL, иначе `false`.

См. также: `ingres_query()`, `ingres_num_fields()`, `ingres_fetch_array()`, `ingres_fetch_object()` и `ingres_fetch_row()`.

ingres_field_length. Получение длины поля

`int` **`ingres_field_length`** `(int index [, resource link])`

Возвращает длину поля (максимальное число байт) с индексом `index` (нумерация начинается с 1) набора записей возвращенных запросом; или `false` при ошибке.

См. документацию: «Ingres/OpenAPI User Guide - Appendix C».

См. также: `ingres_query()`, `ingres_num_fields()`, `ingres_fetch_array()`, `ingres_fetch_object()` и `ingres_fetch_row()`.

ingres_field_precision. Получение точности поля

`int` **`ingres_field_precision`** `(int index [, resource link])`

Возвращает точность поля (только для полей содержащих числовые значения) с индексом `index` (нумерация начинается с 1) набора записей возвращенных запросом.

См. документацию: «Ingres/OpenAPI User Guide - Appendix C».

См. также: `ingres_query()`, `ingres_num_fields()`, `ingres_fetch_array()`, `ingres_fetch_object()` и `ingres_fetch_row()`.

ingres_field_scale. Получение размерности поля

`int` **`ingres_field_scale`** `(int index [, resource link])`

Возвращает масштаб целочисленного поля с индексом `index`.

См. документацию: «Ingres/OpenAPI User Guide - Appendix C».

См. также: `ingres_query()`, `ingres_num_fields()`, `ingres_fetch_array()`, `ingres_fetch_object()` и `ingres_fetch_row()`.

Sybase

sybase_connect. Подключение к серверу Sybase

`int` **`sybase_connect`** `(string servername, string username, string password [, string charset])`

Возвращает дескриптор подключения к серверу Sybase, или `false` при ошибке. Аргумент `servername` должен быть действительным именем, определенном в файле 'interfaces'.

Если функция повторно вызывается с теми же аргументами, новое подключение не создается, а возвращается идентификатор имеющегося.

Подключение к серверу закрывается при завершении скрипта или ранее, если явно вызывается функция `sybase_close()`.

См. также: `sybase_pconnect()`, `sybase_close()`.

sybase_close. Закрытие подключения к Sybase

`bool` **`sybase_close`** `(int link_identifier)`

Возвращает `true`, или `false` при ошибке.

Идентификатор закрываемого подключения указывается в аргументе, если его не указывать, то закрывается последнее, открытое данным скриптом подключение.

Фактически использование данной функции не является обязательным, так как PHP автоматически закрывает все незакрытые неустойчивые подключения при завершении скрипта. Заметьте, устойчивые подключения, созданные функцией `sybase_pconnect()` не закрываются.

См. также: `sybase_connect()`, `sybase_pconnect()`.

sybase_pconnect. Создание устойчивого подключения к серверу Sybase

`int` **`sybase_pconnect`** `(string servername, string username, string password [, string charset])`

Возвращает дескриптор устойчивого подключения, или `false` при ошибке. `sybase_pconnect()` действует подобно `sybase_connect()` с двумя различиями:

Перед подключением, функция пытается проверить имеется ли уже открытое (устойчивое) подключение с параметрами (имя сервера, пользователя и пароль), аналогичными указанным. Если такое подключение обнаруживается, то возвращается его идентификатор, вместо создания нового подключения.

При завершении скрипта, подключение не закрывается, а остается действительным для дальнейшего использования. (Функция `sybase_close()` не может закрыть подключения, созданные с помощью `sybase_pconnect()`).

sybase_select_db. Выбор БД Sybase

```
bool sybase_select_db (string database_name, int link_identifier)
```

Возвращает `true`, или `false` при ошибке.

Устанавливает БД с именем `database_name` активной для текущего подключения или указанного в `link_identifier`. Если подключений не имеется, то косвенно вызывается функция `sybase_connect()` с параметрами по умолчанию.

Последующие запросы, выполняемые функцией, будут адресованы данной БД.

См. также: `sybase_connect()`, `sybase_pconnect()`, и `sybase_query()`

sybase_query. Выполнение запроса к БД Sybase

```
int sybase_query (string query, int link_identifier)
```

Посылает запрос текущей активной БД, для текущего (последнего открытого) подключения или указанного в `link_identifier`. Если подключений не имеется, то косвенно вызывается функция `sybase_connect()` с параметрами по умолчанию.

Возвращает дескриптор набора возвращенных записей, или `false` при ошибке.

См. также: `sybase_select_db()` и `sybase_connect()`.

sybase_affected_rows. Получение числа измененных записей последним запросом

```
int sybase_affected_rows ([int link_identifier])
```

Возвращается число записей, измененных в результате выполнения запросов `DELETE`, `INSERT`, `REPLACE`, или `UPDATE`.

Если последним запросом была команда `DELETE` без ограничения `WHERE`, то из таблицы будут удалены все записи, но эта функция возвратит 0.

Чтобы выяснить, сколько записей было возвращено командой `SELECT`, используйте функцию `sybase_num_rows()`.

Функция доступна только при использовании библиотеки интерфейса CT Sybase, но не DB Sybase.

sybase_num_rows. Получение числа записей возвращенных запросом

```
int sybase_num_rows (int result)
```

См. также: `sybase_query()` и `sybase_fetch_row()`.

sybase_num_fields. Получение числа полей в наборе записей

```
int sybase_num_fields (int result)
```

См. также: sybase_query(), sybase_fetch_field(), sybase_num_rows().

sybase_data_seek. Перемещение внутреннего указателя записи

bool **sybase_data_seek** (int result_identifier, int row_number)

При каждом вызове функции sybase_fetch_row() (или подобной), внутренний курсор записи смещается на следующую. Данная функция позволяет свободно перемещать курсор в наборе записей result_identifier, так, чтобы он указывал на запись с номером row_number (нумерация начинается с 0) при следующем вызове sybase_fetch_row().

Она возвращает true, или false при ошибке.

sybase_result. Получение определенного элемента набора записей

string **sybase_result** (int result, int row, mixed field)

Возвращает содержимое ячейки из набора записей result. В аргументе row указывается номер записи (нумерация начинается с 0), в аргументе field можно указать индекс поля (число), имя поля или полное имя поля (вида: имя_таблицы.имя_поля) или алиас поля (для запросов типа 'SELECT foo AS bar FROM...').

При работе с большими наборами записей более быстро выполняются функции типа _fetch_ (см. ниже). Также заметьте, что численные индексы обрабатываются быстрее строковых.

Не следует вызывать функцию mysql_result() в сочетании с другими функциями обработки набора записей.

Рекомендуется использовать альтернативные функции: sybase_fetch_row(), sybase_fetch_array(), и sybase_fetch_object().

sybase_fetch_row. Занесение записи в нумерованный массив

array **sybase_fetch_row** (int result)

Возвращает массив, соответствующий текущей записи, из набора записей result, возвращенных запросом (последующий вызов функции возвращает следующую запись); или false записей более не имеется. Каждое поле записи сохраняется в нумерованном элементе массива, (нумерация начинается с 0).

См. также: sybase_fetch_array(), sybase_fetch_object(), sybase_data_seek(), sybase_result().

sybase_fetch_array. Занесение записи в массив

array **sybase_fetch_array** (int result)

Возвращает массив, соответствующий текущей записи, из набора записей `result`, возвращенных запросом; или `false` записей более не имеется (при этом курсор записи перемещается на следующую).

Данная функция является расширением функции `sybase_fetch_row()`, и она возвращает объединенный нумерованно-ассоциативный массив. Доступ к элементам массива может осуществляться как через имена полей, так и по их номеру.

Заметьте, что функция не значительно медленнее, чем `sybase_fetch_row()`, но предоставляет дополнительную функциональность.

См. также: `sybase_fetch_row()`

sybase_fetch_object. Получение записи в свойствах объекта

```
int sybase_fetch_object (int result)
```

Возвращает объект, в свойствах которого находятся поля текущей записи; или `false` записей более не имеется.

По скорости (и функциональности) функция идентична `sybase_fetch_array()`, и почти идентична `sybase_fetch_row()`.

См. также: `sybase_fetch_array()` и `sybase_fetch_row()`.

sybase_free_result. Уничтожение набора записей

```
bool sybase_free_result (int result)
```

Функция освобождает память, занимаемую набором записей `result`, возвращенным запросом. Ее следует использовать только, если требуется экономить память, так как память автоматически освобождается при завершении скрипта.

sybase_fetch_field. Получение информации о поле записи в свойствах объекта

```
object sybase_fetch_field (int result [, int field_offset])
```

Если номер поля `field_offset` не указан, при каждом вызове функции возвращаются свойства следующего поля из набора записей `result`.

Возвращаемый объект имеет следующие свойства (и содержит информацию):

- `name` – имя поля. Если поле - результат выполнения функции, имя возвращается в виде `computed#N`, где `#N` – порядковый номер.
- `column_source` – имя таблицы, которой принадлежит поле
- `numeric` – 1, если поле числовое
- `type` – тип поля

См. также: `sybase_field_seek()`

sybase_field_seek. Перемещение к указанному полю

`int` **`sybase_field_seek`** `(int result, int field_offset)`

Перемещается к указанному числом `field_offset` полю. И, если при следующем вызове `sybase_fetch_field ()` не указывается номер поля, будет возвращено это.

См. также: `sybase_fetch_field()`.

sybase_get_last_message. Получение последнего сообщение сервера

`string` **`sybase_get_last_message`** `(void )`

sybase_min_client_severity. Установка минимального уровня требовательности клиента

`void` **`sybase_min_client_severity`** `(int severity)`

Функция доступна только при использовании библиотеки интерфейса CT Sybase, но не DB Sybase.

См. также: `sybase_min_server_severity()`.

sybase_min_server_severity. Установка минимального уровня требовательности сервера

`void` **`sybase_min_server_severity`** `(int severity)`

Функция доступна только при использовании библиотеки интерфейса CT Sybase, но не DB Sybase.

См. также: `sybase_min_client_severity()`.

sybase_min_message_severity. Установка минимального уровня возвращаемых сообщений

`void` **`sybase_min_message_severity`** `(int severity)`

См. также: `sybase_min_error_severity()`.

sybase_min_error_severity. Установка минимального уровня сообщаемых ошибок

`void` **`sybase_min_error_severity`** `(int severity)`

См. также: `sybase_min_message_severity()`.

Oracle

Примерный образец работы с БД:

```
<?php
```

```
putenv("ORACLE_SID=ORACLE");
putenv("ORACLE_HOME=/opt/oracle/oracle/8.0.3");

$conn = ora_login("user_nam", "passwd");
// $conn = ora_login("user_nam@server", "passwd"); // или так
$curs = ora_open($conn);

ora_commitoff($conn);

$query = "select * from t";

ora_parse($curs, $query);
ora_exec($curs);

// ora_do($conn, $query);  // краткая версия запроса

$ncols = ora_numcols($curs); // число полей
$nrows = ora_numrows($curs); // число записей

$hdr=true;
while (ora_fetch($curs)) { // следующая запись
   if($hdr)   // выводить заголовок только в начале
      for ($i=0; $i<$ncols; $i++)
         printf("[$i] %s (%s); \n", ora_columnname($curs, $i),
               ora_columntype($curs, $i));
   $hdr=false;
   for ($i=0; $i<$ncols; $i++)
      print ora_getcolumn($curs,$i). "; "; // поле
   print "\n";
}

?>
```

Ora_Logon. Подключение к серверу Oracle

`int` **`ora_logon`** `(string user, string password)`

Возвращает дескриптор подключения к БД, или `false` при ошибке.

Подключения также могут создаваться с использованием синтаксиса SQL*Net указывая TNS имя пользователя `user` в виде:

```
$conn = Ora_Logon( "system/user@host" , "pass" );
```

Если имеются не-ASCII символы, то должна быть установлена (до запуска сервера) переменная окружения NLS_LANG.

Ora_Logoff. Закрытие подключения

`int` **`ora_logoff`** `(int connection)`

Возвращает `true`, или `false` при ошибке.

См. также: `ora_logon()`.

Ora_pLogon. Создание устойчивого подключения к серверу Oracle

`int` **`ora_plogon`** `(string user, string password)`

См. также: `ora_logon()`.

Ora_Open. Открытие курсора

`int` **`ora_open`** `(int connection)`

Возвращает индекс курсора для подключения `connection`, или `false` при ошибке.

Ora_Close. Закрытие курсора

`int` **`ora_close`** `(int cursor)`

Закрывает курсор, открытый `ora_open()`. Возвращает `true`, или `false` при ошибке.

Ora_CommitOn. Разрешение автовыполнения запросов

`int` **`ora_commiton`** `(int conn)`

Каждый запрос для подключения `conn`, выполняемый функцией `ora_exec()` будет автоматически завершаться. Возвращает `true`, или `false` при ошибке.

Ora_CommitOff. Запрет автовыполнения запросов

`int` **`ora_commitoff`** `(int conn)`

Ora_Commit. Завершение текущей транзакции

`int` **`ora_commit`** `(int conn)`

Транзакцией является серия изменений в БД (запросов), после последнего завершения транзакции (если автозавершение было отключено). Возвращает `true`, или `false` при ошибке.

Ora_Rollback. Отмена транзакции

`int` **`ora_rollback`** `(int connection)`

Функция, обратная `ora_commit()`. Возвращает `true`, или `false` при ошибке.

Ora_Parse. Интерпретация SQL запроса

`int` **`ora_parse`** `(int cursor_ind, string sql_statement, int defer)`

Подготавливает для исполнения запрос SQL или блок PL/SQL и связывает с курсором. Возвращает 0, или -1 при ошибке.

См. также: `ora_exec()`, `ora_fetch()`, и `ora_do()`.

Ora_Exec. Исполнение подготовленного запроса

`int` **`ora_exec`** `(int cursor)`

Возвращает `true`, или `false` при ошибке.

См. также: `ora_parse()`, `ora_fetch()`, и `ora_do()`.

Ora_Do. Одновременное выполнение команд Parse, Exec, Fetch

`int` **`ora_do`** `(int conn, string query)`

Функция является комбинацией трех функций `ora_parse()`, `ora_exec()`, `ora_fetch()`. Возвращает `true`, или `false` при ошибке.

См. также: `ora_parse()`, `ora_exec()`, и `ora_fetch()`.

Ora_Bind. Связать переменную PHP с SQL параметром Oracle

`int` **`ora_bind`** `(int cursor, string PHP_var_name, string SQL_param_name, int length [, int type])`

Возвращает `true`, или `false` при ошибке.

SQL параметр должен указываться в виде "`:name`». Необязательным аргументом `type` можно указать вид SQL параметра (входящий, выходящий): in/out (0, по умолчанию), in (1) или out (2). В PHP 3.0.1, можно использовать вместо цифр константы: ORA_BIND_INOUT, ORA_BIND_IN и ORA_BIND_OUT.

Функция должна вызываться после `ora_parse()` и до `ora_exec()`.

```
<?php
ora_parse($curs, "declare tmp INTEGER; ".
         "begin tmp := :in; :out := tmp; :x := 7.77; end;");
ora_bind($curs, "result", ":x", $len, 2);
ora_bind($curs, "input", ":in", 5, 1);
ora_bind($curs, "output", ":out", 5, 2);
$input = 765;
ora_exec($curs);
echo "Result: $result; Out: $output; In: $input";
?>
```

Ora_Error. Получение сообщения об ошибке Oracle

`string` **`Ora_Error`** `(int cursor_or_connection)`

Сообщение возвращается строек вида: `XXX-NNNNN`, где `XXX` означает место возникновения ошибки, а `NNNNN` - причину.

Ora_ErrorCode. Получение кода ошибки Oracle

`int` **`Ora_ErrorCode`** `(int cursor_or_connection)`

Возвращает численное значение ошибки произошедшей при последней операции.

Значение 0 означает отсутствие ошибки.

Ora_Numcols. Число полей возвращенных запросом

`int` **`ora_numcols`** `(int cursor_ind)`

См. также: `ora_parse()`, `ora_exec()`, `ora_fetch()`, и `ora_do()`.

Ora_Numrows. Число записей возвращенных запросом

`int` **`ora_numrows`** `(int cursor_ind)`

Ora_Fetch. Получение записи

`int` **`ora_fetch`** `(int cursor)`

Возвращает `true`, или `false` при ошибке или если записей больше не имеется.

См. также: `ora_parse()`,`ora_exec()`, и `ora_do()`.

Ora_Fetch_Into. Получение записи в массиве

`int` **`ora_fetch_into`** `(int cursor, array result [, int flags])`

```
<?php
array($results);
ora_fetch_into($cursor, &$results);
echo $results[0], $results[1];
?>
```

Заметьте, что массив необходимо передавать по ссылке.

См. также: `ora_parse()`, `ora_exec()`, `ora_fetch()`, и `ora_do()`.

Ora_GetColumn. Получение данных поля записи

`mixed` **`ora_getcolumn`** `(int cursor, mixed column)`

Возвращает значение поля (или результата SQL/PL функции), или `false` при ошибке. Заметьте, что возвращаемые данные могут иметь значения: NULL, "", 0, «0».

Ora_ColumnName. Получение имени поля

```
string Ora_ColumnName (int cursor, int column)
```

Возвращает имя в верхнем регистре.

Ora_ColumnSize. Получение размера поля

```
int Ora_ColumnSize (int cursor, int column)
```

Ora_ColumnType. Получение типа поля

```
string Ora_ColumnType (int cursor, int column)
```

Возвращаемая строка может иметь следующие значения:

```
"VARCHAR2"
"VARCHAR"
"CHAR"
"NUMBER"
"LONG"
"LONG RAW"
"ROWID"
"DATE"
"CURSOR"
```

Oracle 8

Эта группа функций позволяет обращаться к серверам БД Oracle8 и Oracle7, используя Oracle8 Call-Interface (OCI8). Для этого требуются клиентские библиотеки Oracle8.

Это расширение более функционально, нежели стандартное; оно поддерживает связывание переменных PHP с переменными Oracle, имеет полную поддержку типов LOB, FILE, ROWID и позволяет использовать определяемые пользователем переменные.

До использования этого расширения необходимо установить параметры окружения Oracle. Это включает установку следующих системных переменных:

- `ORACLE_HOME`
- `ORACLE_SID`
- `LD_PRELOAD`
- `LD_LIBRARY_PATH`
- `NLS_LANG`
- `ORA_NLS33`

После этого необходимо добавить пользователя (обычно nobody или www), от имени которого запускается PHP (веб сервер) в группу пользователей oracle.

OCIServerVersion. Получение строки информации сервера

```
string OCIServerVersion (int conn)
```

```
<?php
   $conn = OCILogon("scott","tiger");
   print "Server Version: " . OCIServerVersion($conn);
   OCILogOff($conn);
?>
```

OCILogon. Подключение к серверу Oracle

```
int OCILogon (string username, string password [, string db])
```

Возвращает дескриптор подключения необходимый для указания для большинства функций OCI. В третьем аргументе можно указать либо константное имя локальной БД, либо ссылку на раздел в файле tnsnames.ora; если этот аргумент не указан, PHP будет использовать переменные окружения ORACLE_SID или TWO_TASK (tnsnames.ora) для того, чтобы определить, к какой БД следует подключиться.

Подключения, созданные этой функцией становятся разделяемыми на уровне скрипта, что означает, что все команды commit и rollback будут применяться ко всем транзакциям, даже если они принадлежат разным подключениям (см. пример ниже).

См. также: `OCIPLogon()` и `OCINLogon()`.

OCINLogon. Создание изолированного подключения к БД Oracle

```
int OCINLogon (string username, string password [, string db])
```

Функция сходна с `OCILogon()`, но отличается тем, что все операции commit и rollback применяются только к транзакциям указанного подключения.

Пример демонстрирует разделение подключений.

```
<?php
$db = "";

$c1 = ocilogon("scott","tiger",$db);
//$c2 = ocilogon("scott","tiger",$db);  // для сравнения
$c2 = ocinlogon("scott","tiger",$db);

function create_table($conn)
{ $stmt = ociparse($conn,"create table scott.hallo (test
varchar2(64))");
  ociexecute($stmt);
  echo $conn." Таблица создана \n\n";
}
```

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

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

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

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

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

function select_data($conn)
{ $stmt = ociparse($conn,"select * from scott.hallo");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn."----selecting\n\n";
  while (ocifetch($stmt))
    echo $conn." <".ociresult($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);
?>
```

См. также: `OCILogon()` и `OCIPLogon()`.

OCILogOff. Отключение от Oracle

`int` **`OCILogOff`** `(int connection)`

OCIPLogon. Создание устойчивого подключения к Oracle

`int` **`OCIPLogon`** `(string username, string password [, string db])`

Функция сходна с `OCILogon()`, но отличается тем, что подключение не закрывается после завершения скрипта и остается действительным для дальнейшего использования. См. также: `OCILogon()` и `OCINLogon()`.

OCIInternalDebug. Разрешение или запрещение внутренней отладки

`void` **`OCIInternalDebug`** `(int onoff)`

По умолчанию отладка запрещена. Указание в аргументе значения 0 выключает отладку, а 1 - включает.

OCIDefineByName. Назначение PHP переменной для возвращения значения из запроса SELECT

`int` **`OCIDefineByName`** `(int stmt, string ColumnName, mixed variable [, int type])`

Используется, когда необходимо чтобы запрос выборки `stmt` возвращал данные поля `ColumnName`, в переменную `variable` (тип возвращаемых данных `type`

можно не указывать). Имя поля должно быть указано в верхнем регистре (в запросе регистр не важен). Если вы определяете переменную, отсутствующую в запросе, ошибка не возникает.

Типы абстрактных данных (LOB/ROWID/BFILE) необходимо предварительно инициализировать функцией `OCINewDescriptor()`. См. также: `OCIBindByName()`.

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

$stmt = OCIParse($conn,"select empno, ename from emp");

/* the define MUST be done BEFORE ociexecute! */

OCIDefineByName($stmt,"EMPNO",$empno);
OCIDefineByName($stmt,"ENAME",$ename);

OCIExecute($stmt);

while (OCIFetch($stmt))
    echo "empno:".$empno.", ename:".$ename."\n";

OCIFreeStatement($stmt);
OCILogoff($conn);
?>
```

OCIBindByName. Связывает переменную PHP с Oracle

`int` **`OCIBindByName`** `(int stmt, string ph_name, mixed &variable, int length [, int type])`

Назначает переменную PHP `variable` для ввода-вывода данных при операциях с Oracle в качестве параметра `ph_name`. Аргумент `length` устанавливает размер данных переменной в байтах; значение -1 указывает, что размер определяется автоматически.

Для операций с абстрактными типами (LOB/ROWID/BFILE) предварительно необходимо инициализировать их дескриптор функцией `OCINewDescriptor()`. Для этих типов размер `length` должен быть указан значением -1. Тип `type` может определятся следующими константами: `OCI_B_FILE` (двоичный файл), `OCI_B_CFILE` (символьный файл), `OCI_B_CLOB` (символьный LOB), `OCI_B_BLOB` (двоичный LOB) и `OCI_B_ROWID` (ROWID).

```
<?php
$sth = OCIParse ( $dbh, "begin sp_newaddress( :address_id, '$firstname',
 '$lastname', '$company', '$address1', '$address2', '$city', '$state',
 '$postalcode', '$country', :error_code );end;" );
```

```
// Вызвать хранимую процедуру sp_newaddress, с параметрами :address_id
// (входной-выходной параметр) и :error_code being (выходной).

   OCIBindByName ( $sth, ":address_id", $addr_id, 10 );
   OCIBindByName ( $sth, ":error_code", $errorcode, 10 );
   OCIExecute ( $sth );
?>

//****************************************
<?php     $conn = OCILogon("scott","tiger");

$stmt = OCIParse($conn,"insert into emp (empno, ename) ".
         "values (:empno,:ename) returning ROWID into :rid");

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

$rowid = OCINewDescriptor($conn,OCI_D_ROWID);

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

$update = OCIParse($conn,"update emp set sal = :sal
                          where ROWID = :rid");
OCIBindByName($update,":rid",&$rowid,-1,OCI_B_ROWID);
OCIBindByName($update,":sal",&$sal,32);

$sal = 10000;

while (list($empno,$ename) = each($data)) {
   OCIExecute($stmt);
   OCIExecute($update);
}

$rowid->free();
OCIFreeStatement($update);
OCIFreeStatement($stmt);

$stmt = OCIParse($conn,"select * from emp
                       where empno in (1111,2222,3333)");
OCIExecute($stmt);
while (OCIFetchInto($stmt,&$arr,OCI_ASSOC))
   var_dump($arr);

OCIFreeStatement($stmt);

$stmt = OCIParse($conn,"delete from emp
```

```
                    where empno in (1111,2222,3333)");
OCIExecute($stmt);
OCIFreeStatement($stmt);

OCILogoff($conn);
?>
```

OCIParse. Подготовка запроса к выполнению

`int` **`OCIParse`** `(int conn, strint query)`

Возвращает дескриптор запроса `query` для подключения `conn`; или `false` при ошибке.

OCIExecute. Выполнение запроса

`int` **`OCIExecute`** `(int statement [, int mode])`

Выполняет запрос, предварительно подготовленный `OCIParse()`. Аргументом `mode` можно указать режим исполнения (по ум. `OCI_COMMIT_ON_SUCCESS`); если не требуется автоматическое выполнения запроса, указывайте `OCI_DEFAULT`.

OCICommit. Завершение незавершенных транзакций

`int` **`OCICommit`** `(int connection)`

OCIRollback. Отмена незавершенных транзакций

`int` **`OCIRollback`** `(int connection)`

OCINewDescriptor. Инициализация дескриптора LOB/FILE

`string` **`OCINewDescriptor`** `(int connection [, int type])`

Выделяет ресурсы, необходимые для хранения дескрипторов или LOB локаторов. Тип можно указать в аргументе `type`: `OCI_D_FILE`, `OCI_D_LOB` (по ум.), `OCI_D_ROWID`. Для объектов LOB, доступны методы `load`, `save`, и `savefile`, а для BFILE – только `load`.

```
<?php       // фрагмент for retrieve data use (after fetch):

$result = OCIResult($stmt, $n);
if (is_object ($result)) $result = $result->load();

// For INSERT or UPDATE statement use:

$sql = "insert into table (field1, field2) values (field1 = 'value',
             field2 = empty_clob()) returning field2 into :field2";
OCIParse($conn, $sql);
```

```
$clob = OCINewDescriptor($conn, OCI_D_LOB);
OCIBindByName ($stmt, ":field2", &$clob, -1, OCI_B_CLOB);
OCIExecute($stmt, OCI_DEFAULT);
$clob->save ("some text");
?>

<?php  // Скрипт вызывается формой HTML, передавая параметры:
       //  $user, $password, $table, $where, и $commitsize.
    $conn = OCILogon($user, $password);
    $stmt = OCIParse($conn,"select rowid from $table $where");
    $rowid = OCINewDescriptor($conn,OCI_D_ROWID);
    OCIDefineByName($stmt,"ROWID",&$rowid);
    OCIExecute($stmt);
    while ( OCIFetch($stmt) ) {
       $nrows = OCIRowCount($stmt);
       $delete = OCIParse($conn,"delete from $table
                              where ROWID = :rid");
       OCIBindByName($delete,":rid",&$rowid,-1,OCI_B_ROWID);
       OCIExecute($delete);
       print "$nrows\n";
       if ( ($nrows % $commitsize) == 0 ) {
           OCICommit($conn);
       }
    }
    $nrows = OCIRowCount($stmt);
    print "$nrows удалено...\n";
    OCIFreeStatement($stmt);
    OCILogoff($conn);
?>


<?php  /* Скрипт загружает файл в поле LOB */
  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">
</form>
<?php
  } else {
     // $lob_upload contains the temporary filename of the uploaded file
     $conn = OCILogon($user, $password);
     $lob = OCINewDescriptor($conn, OCI_D_LOB);
     $stmt = OCIParse($conn,"insert into $table (id, the_blob)
                 values(my_seq.NEXTVAL, EMPTY_BLOB())
               returning the_blob into :the_blob");
     OCIBindByName($stmt, ':the_blob', &$lob, -1, OCI_B_BLOB);
```

```
    OCIExecute($stmt);
    if($lob->savefile($lob_upload)){
       OCICommit($conn);
       echo "Blob successfully uploaded\n";
    }else{
       echo "Couldn't upload Blob\n";
    }
    OCIFreeDesc($lob);
    OCIFreeStatement($stmt);
    OCILogoff($conn);
  }
?>
```

OCIRowCount. Получение числа измененных записей

```
int OCIRowCount (int statement)
```

Возвращается число записей измененных запросом (например, update). Функция не возвращает число возвращенных запросом записей!

```
<?php
    $conn = OCILogon("scott","tiger");
    $stmt = OCIParse($conn,"create table emp2 as select * from emp");
    OCIExecute($stmt);
    print OCIRowCount($stmt) . " записей добавлено.<BR>";
    OCIFreeStatement($stmt);
    $stmt = OCIParse($conn,"delete from emp2");
    OCIExecute($stmt);
    print OCIRowCount($stmt) . " записей удалено.<BR>";
    OCICommit($conn);
    OCIFreeStatement($stmt);
    $stmt = OCIParse($conn,"drop table emp2");
    OCIExecute($stmt);
    OCIFreeStatement($stmt);
    OCILogOff($conn);
?>
```

OCINumCols. Получение числа полей в запросе

```
int OCINumCols (int stmt)
```

```
<?php
    $conn = OCILogon("scott", "tiger");
    $stmt = OCIParse($conn,"select * from emp");
    OCIExecute($stmt);
    while ( OCIFetch($stmt) ) {
        $ncols = OCINumCols($stmt);
        for ( $i = 1; $i <= $ncols; $i++ ) {
```

```
            $column_name  = OCIColumnName($stmt,$i);
            $column_value = OCIResult($stmt,$i);
            print $column_name . ': ' . $column_value . "\n";
        }
        print "\n";
    }
    OCIFreeStatement($stmt);
    OCILogoff($conn);
?>
```

OCIResult. Получение значения поля записи возвращенной запросом

`mixed` **`OCIResult`** `(int statement, mixed column)`

Возвращает данные поля `column` текущей записи (см. `OCIFetch()`). Возвращает все типы данных как строки, за исключением абстрактных типов (ROWID, LOB и FILE).

OCIFetch. Занесение следующей возвращенной записи в буфер результата

`int` **`OCIFetch`** `(int statement)`

Позволяет получить запись после запроса выборки (SELECT) с помощью последующих вызовов `OCIResult()`, возвращающих поля записи.

OCIFetchInto. Занесение следующей записи в массив

`int` **`OCIFetchInto`** `(int stmt, array &result [, int mode])`

Функция позволяет поочередно получить все записи, возвращенные запросом выборки. По ум. массив `result` будет содержать все поля записи (содержащие данные не равные NULL) с нумерацией начиная с 1.

Аргумент `mode` позволяет изменить действие функции. Константы могут комбинироваться сложением (например, `OCI_ASSOC+OCI_RETURN_NULLS`):

- `OCI_ASSOC` – возвращать ассоциативный массив.
- `OCI_NUM` – возвращать нумерованный массив. (по ум.)
- `OCI_RETURN_NULLS` – возвращать пустые поля.
- `OCI_RETURN_LOBS` – возвращать содержимое полей LOB вместо их дескриптора.

OCIFetchStatement. Занесение результата запроса в массив

`int` **`OCIFetchStatement`** `(int stmt, array &variable)`

Возвращает число записей занесенных в массив `variable`.

```
<?php
```

```
$conn = OCILogon("scott","tiger");
$stmt = OCIParse($conn,"select * from emp");
OCIExecute($stmt);

$nrows = OCIFetchStatement($stmt,$results);
if ( $nrows > 0 ) {
   print "<TABLE BORDER=\"1\"><TR>\n";
   while ( list($key, $val) = each($results) )
      print "<TH>$key</TH>\n";
   print "</TR>\n";

   for ( $i = 0; $i < $nrows; $i++ ) {
      print "<TR>\n";
      reset($results);
      while ( $column = each($results) ) {
         $data = $column['value'] [$i];
         print "<TD>$data</TD>\n";
      }
      print "</TR>\n";
   }
   print "</TABLE>\n";
}
print "Получено $nrows записей <BR>\n";

OCIFreeStatement($stmt);
OCILogoff($conn);
?>
```

OCIColumnIsNULL. Проверка содержит ли поле записи значение NULL

`int` **`OCIColumnIsNULL`** `(int stmt, mixed column)`

Возвращает `true`, если поле `column` в результате запроса `stmt` имеет значение NULL. Поле может указываться номером (начиная с 1) или именем.

OCIColumnName. Получение имени поля

`string` **`OCIColumnName`** `(int stmt, int col)`

```
<?php
    $conn = OCILogon("scott", "tiger");
    $stmt = OCIParse($conn,"select * from emp");
    OCIExecute($stmt);
    print "<TABLE BORDER=\"1\"><TR><TH>Name</TH><TH>Type</TH>".
                  "<TH>Length</TH></TR>";
    $ncols = OCINumCols($stmt);
    for ( $i = 1; $i <= $ncols; $i++ ) {
        $column_name  = OCIColumnName($stmt,$i);
```

```
        $column_type  = OCIColumnType($stmt,$i);
        $column_size  = OCIColumnSize($stmt,$i);
        print "<TR><TD>$column_name</TD><TD>$column_type</TD>";
        print "<TD>$column_size</TD></TR>";
    }
    OCIFreeStatement($stmt);
    OCILogoff($conn);
?>
```

См. также: `OCINumCols()`, `OCIColumnType()`, и `OCIColumnSize()`.

OCIColumnSize. Получение размера поля

`int` **`OCIColumnSize`** `(int stmt, mixed column)`

См. также: `OCINumCols()`, `OCIColumnName()`, и `OCIColumnSize()`.

OCIColumnType. Получение типа поля

`mixed` **`OCIColumnType`** `(int stmt, int col)`

См. также: `OCINumCols()`, `OCIColumnName()`, и `OCIColumnSize()`.

OCIStatementType. Получение типа OCI запроса

`string` **`OCIStatementType`** `(int stmt)`

Возвращает одно из следующих значений:

1. «SELECT»
2. «UPDATE»
3. «DELETE»
4. «INSERT»
5. «CREATE»
6. «DROP»
7. «ALTER»
8. «BEGIN»
9. «DECLARE»
10. «UNKNOWN»

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

    $stmt = OCIParse($conn,$sql);
    if ( OCIStatementType($stmt) == "DELETE" ) {
```

```
        die "Вам не разрешено удалять записи этой таблицы";
    }

    OCILogoff($conn);
?>
```

OCINewCursor. Получение нового курсора (дескриптора запроса)

```
int OCINewCursor (int conn)
```

Используется для связывания ссылочных курсоров.

```
<?php // хранимая процедура info.output возвращает ref cursor в :data

$conn = OCILogon("scott","tiger");
$curs = OCINewCursor($conn);
$stmt = OCIParse($conn,"begin info.output(:data); end;");

ocibindbyname($stmt,"data",&$curs,-1,OCI_B_CURSOR);
ociexecute($stmt);
ociexecute($curs);

while (OCIFetchInto($curs,&$data)) {
    var_dump($data);
}

OCIFreeCursor($stmt);
OCIFreeStatement($curs);
OCILogoff($conn);
?>

<?php   // Example 2. REF CURSOR в запросе select
$conn = OCILogon("scott","tiger");
$count_cursor = "CURSOR(select count(empno) num_emps from emp " .
                "where emp.deptno = dept.deptno) as EMPCNT from dept";
$stmt = OCIParse($conn,"select deptno,dname,$count_cursor");

ociexecute($stmt);
print "<TABLE BORDER=\"1\"><TR><TH>DEPT NAME</TH>";
print "<TH>DEPT #</TH><TH># EMPLOYEES</TH></TR>";

while (OCIFetchInto($stmt,&$data,OCI_ASSOC)) {
    print "<TR>";
    $dname  = $data["DNAME"];
    $deptno = $data["DEPTNO"];
    print "<TD>$dname</TD><TD>$deptno</TD>";
    ociexecute($data[ "EMPCNT" ]);
    while (OCIFetchInto($data[ "EMPCNT" ],&$subdata,OCI_ASSOC)) {
```

```
        $num_emps = $subdata["NUM_EMPS"];
        print  "<TD>$num_emps</TD>";
    }
    print "</TR>";
}
print "</TABLE>";
OCIFreeStatement($stmt);
OCILogoff($conn);
?>
```

OCIFreeStatement. Освобождение ресурсов занимаемых запросом

`int` **`OCIFreeStatement`** `(int stmt)`

Возвращает `true`, или `false` при ошибке.

OCIFreeCursor. Освобождение ресурсов курсора

`int` **`OCIFreeCursor`** `(int stmt)`

Возвращает `true`, или `false` при ошибке.

OCIFreeDesc. Уничтожение дескриптора большого объекта

`int` **`OCIFreeDesc`** `(object lob)`

Возвращает `true`, или `false` при ошибке.

OCIError. Получение последнего сообщения об ошибке

`array` **`OCIError`** `([int stmt|conn|global])`

Возвращает ассоциативный массив, содержащий элементы: `code` (номер ошибки) и `message` (сообщение ошибки). Если ошибки не произошло, возвращает `false`.

Аргументом можно указать дескриптор, для которого получается сообщение.

dBase

Данная группа функций позволяет работать с данными БД dBase (dbf).

Индексы и memo поля не поддерживаются. Блокировка также не возможна, поэтому, если два процесса веб-сервера одновременно модифицируют один файл dBase, это может привести к его порче.

В отличие от SQL БД, структура «БД» dBase не поддается модификации. dBase файл представляет собой простую последовательность записей фиксированной длины; добавляемые записи дописываются в конец файла, а удаленные сохраняются до вызова функции `dbase_pack()`.

Рекомендуется использование современных БД; например, MySQL или Postgres. Поэтому поддержка dBase в PHP предназначена исключительно для экспорта и импорта (для облегчения обмена с другими приложениями).

dbase_create. Создание БД dBase

```
int dbase_create (string filename, array fields)
```

Аргумент `fields` это массив массивов, где каждый из массивов описывает одно поле БД (таблицы). Каждое поле состоит из имени, символа обозначающего тип, и (при необходимости) размера поля и его точности.

Возможные типы полей:

- L – двоичное, не имеет точности или размера.
- M - Memo. Не поддерживается PHP, не имеет точности или размера.
- D – Дата, сохраняется как ГГГГММДД, не имеет точности или размера.
- N – Число, имеет длину и точность (число знаков после запятой).
- C – Строка.

Если файл БД был успешно создан, возвращается его дескриптор; иначе `false`.

```
$dbname = "/tmp/test.dbf";  // "database" name

$def =           // database "definition"
    array(
        array("date",     "D"),
        array("name",     "C",  50),
        array("age",      "N",   3, 0),
        array("email",    "C", 128),
        array("ismember", "L")
    );

if (!dbase_create($dbname, $def))    print "Error!";
```

dbase_open. Открытие БД dBase

```
int dbase_open (string filename, int flags)
```

В аргументе `flags` указывается режим открытия (0 – только для чтения, 1 – только для записи, и 2 – для чтения и записи). Возвращает дескриптор БД; или `false` при ошибке.

dbase_close. Закрытие БД dBase

```
bool dbase_close (int dbase_identifier)
```

dbase_pack. Обновление БД dBase

`bool` **`dbase_pack`** `(int dbase_identifier)`

При этом удаляются все записи, помеченные с помощью `dbase_delete_record()`.

dbase_add_record – добавление записи

`bool` **`dbase_add_record`** `(int dbase_identifier, array record)`

Добавляет запись, поля которой содержаться в массиве `record` в БД. Если число полей записи не соответствует числу полей в БД, добавление не производится и функция возвращает `false`.

dbase_replace_record. Замена записи

`bool` **`dbase_replace_record`** `(int dbase_identifier, array record, int dbase_record_number)`

Заменяет запись (с номером `dbase_record_number`), поля которой содержаться в массиве `record`. Если число полей записи не соответствует числу полей в БД, добавление не производится и функция возвращает `false`.

Номера записей лежат в пределах от 1 до значения, возвращенного `dbase_numrecords()`.

dbase_delete_record. Удаление записи

`bool` **`dbase_delete_record`** `(int dbase_identifier, int record)`

Помечает запись с номером `record` для удаления последующим вызовом `dbase_pack()`.

dbase_get_record. Получение записи

`array` **`dbase_get_record`** `(int dbase_identifier, int record)`

Возвращает массив, содержащий поля записи с номером `record`. Массив индексируется начиная с 0, и имеет элемент с индексом 'deleted' равный 1, если запись была удалена (см. `dbase_delete_record()`. Каждое поле преобразуется у соответствующему типу PHP. (Даты сохраняются как строки).

dbase_get_record_with_names. Занесение записи в ассоциативный массив

`array` **`dbase_get_record_with_names`** `(int dbase_identifier, int record)`

Функция аналогична `dbase_get_record()`, но вместо числовых индексов используются имена полей.

dbase_numrecords. Число записей БД

```
int dbase_numrecords (int dbase_identifier)
```

При обращении к записям БД, их возможные номера лежат в пределах от 1 до значения, возвращаемого данной функцией.

dbase_numfields. Число полей БД

```
int dbase_numfields (int dbase_identifier)
```

При обращении к полям записи, их возможные номера лежат в пределах от 0 до значения, возвращаемого данной функцией, уменьшенного на единицу.

```
$rec = dbase_get_record($db, $recno);
$nf  = dbase_numfields($db);
for ($i=0; $i < $nf; $i++) {
    print $rec[$i]."<br>\n";
}
```

DBM

Эта группа функций позволяет манипулировать записями в dbm-БД. Этот тип БД поддерживается системными библиотеками: Berkeley DB, GDBM, наряду со встроенной поддержкой «плоских файлов». Данные в них сохраняются в виде пар «имя/значение».

```
$dbm = dbmopen ("lastseen", "w");
if (dbmexists ($dbm, $userid)) {
    $last_seen = dbmfetch ($dbm, $userid);
} else {
    dbminsert ($dbm, $userid, time());
}
do_stuff();
dbmreplace ($dbm, $userid, time());
dbmclose ($dbm);
```

dbmopen. Открытие БД DBM

```
int dbmopen (string filename, string flags)
```

Первый аргумент содержит полное имя DBM файла БД, а второй - режим открытия: «r», «w», «n», «c» соответственно: для чтения, для чтения и записи, создание новой БД (для чтения и записи), добавление в имеющуюся БД или ее создание.

Возвращает дескриптор БД, используемый всеми последующими функциями, или `false` при ошибке.

При использовании поддержки NDBM, NDBM фактически создает файлы `filename.dir` и `filename.pag`. GDBM (как и встроенная библиотека поддержки

«плоских файлов») использует только один файл, а Berkeley DB создает файл `filename.db`. Заметьте, что PHP самостоятельно блокирует файл, в дополнение к тому, что он может блокироваться библиотекой DBM. PHP не удаляет файлы `.lck`, которые создает.

dbmclose. Закрытие БД dbm

`bool` **`dbmclose`** `(int dbm_identifier)`

Разблокирует и закрывает открытую БД.

dbmexists. Проверка существования имени

`bool` **`dbmexists`** `(int dbm_identifier, string key)`

Возвращает `true`, если имеется значение с именем `key`; или `false` при ошибке.

dbmfetch. Получение значения элемента

`string` **`dbmfetch`** `(int dbm_identifier, string key)`

Возвращает значение для имени `key`.

dbminsert. Добавление элемента

`int` **`dbminsert`** `(int dbm_identifier, string key, string value)`

Возвращает –1, если БД была открыта только для чтения; 0, если добавление было успешно выполнено; и 1, если указанный элемент уже существует (для замены используйте `dbmreplace()`).

dbmreplace. Замена значения

`bool` **`dbmreplace`** `(int dbm_identifier, string key, string value)`

Если указанного элемента `key` не существовало, он создается.

dbmdelete. Удаление элемента

`bool` **`dbmdelete`** `(int dbm_identifier, string key)`

Возвращает `false`, если элемент `key` в БД не присутствует.

dbmfirstkey. Получение имени первого элемента

`string` **`dbmfirstkey`** `(int dbm_identifier)`

Порядок элементов не гарантируется (так как могут использоваться внутренние хеш таблицы).

dbmnextkey. Получение следующего элемента

```
string dbmnextkey (int dbm_identifier, string key)
```

В примере демонстрируется, как можно получить все значения БД:

```
$key = dbmfirstkey ($dbm_id);
while ($key) {
    echo "$key = " . dbmfetch ($dbm_id, $key) . "\n";
    $key = dbmnextkey ($dbm_id, $key);
}
```

dblist. Получение описания используемой библиотеки DBM

```
string dblist (void)
```

Абстракция уровня dbm

Эти функции предназначены для работы с разновидностями Berkeley DB. Фактически, поддержка ограничена современными подобиями Sleepycat Software DB2 support. (не путайте с IBM DB2, использующей интерфейс ODBC).

Действие некоторых функций различается в зависимости от реализации БД. Например, функции `dba_optimize()` и `dba_sync()` действуют согласно описанию, но могут быть совершенно бесполезны для некоторых видов БД.

Чтобы добавить поддержку ниже перечисленных БД в PHP используйте указанные опции --with скрипта конфигурирования configure:

- Dbm – начальная версия Berkeley DB. Постарайтесь ее не использовать из-за проблем совместимости (--with-dbm).
- Ndbm – новый и более удобный тип, нежели dbm. Но имеет многие недостатки dbm (--with-ndbm).
- Gdbm – GNU database manager (ftp://ftp.gnu.org/pub/gnu/gdbm/). (--with-gdbm)
- DB2 – Sleepycat Software's DB2 (http://www.sleepycat.com/). Описывается как: «Высокоскоростная, подходящая как для клиентских, так и для клиент-серверных приложений». Использует бинарные деревья, вместо хеш таблиц. (--with-db2)
- DB3 – Sleepycat Software's DB3. (--with-db3)
- Cdb – «быстрая, облегченная, для хранения констант»; предоставляет доступ только для чтения. (--with-cdb)

```
<?php

$id = dba_open ("/tmp/test.db", "n", "db2");

if (!$id) {
```

```
    echo "dba_open failed\n";
    exit;
}

dba_replace ("key", "This is an example!", $id);

if (dba_exists ("key", $id)) {
    echo dba_fetch ("key", $id);
    dba_delete ("key", $id);
}
dba_sync($id);
dba_close($id);
?>
```

DBA позволяет работать с двоичными данными, и ограничена только возможностями БД. БД, использующие файлы должны предоставлять возможность указания режима открытия файла БД (или его создания), и обычно это указывается в аргументе функций `dba_open()` и `dba_popen()`.

Получить последовательно все содержащиеся в БД элементы можно используя функции `dba_firstkey()` и `dba_nextkey()`. Не изменяйте БД во время просмотра.

```
<?php

# ...open database...

$key = dba_firstkey ($id);

while ($key != false) {
    if (...) {      # запомнить имя для полседующей операции
        $handle_later[] = $key;
    }
    $key = dba_nextkey ($id);
}

for ($i = 0; $i < count($handle_later); $i++)
    dba_delete ($handle_later[$i], $id);

?>
```

dba_open. Открытие БД

`int` **`dba_open`** `(string path, string mode, string handler [, ...])`

Открывает БД типа `handler`, содержащуюся в файле `path` в режиме `mode`, который может принимать следующие значения: «r» для чтения, «w» для чтения и записи, «c» для открытия уже существующей (или создания отсутствующей БД) с

доступом для чтения и записи, и «n» для создания файла БД заново и открытия его для чтения и записи. Дополнительные аргументы могут указываться при необходимости, для некоторых БД. Функция возвращает дескриптор открытой БД (необходимый последующим функциям для выполнения операций с БД); или, при ошибке, `false`.

См. также: `dba_popen()`, `dba_close()`

dba_close. Закрытие БД

`void` **`dba_close`** `(int handle)`

См. также: `dba_open()` и `dba_popen()`

dba_popen. Устойчивое открытие БД

`int` **`dba_popen`** `(string path, string mode, string handler [, ...])`

Функция подобна `dba_open()`, но подключение не закрывается при завершении скрипта, а остается действительным для последующего использования.

См. также: `dba_close()`

dba_exists. Проверка наличия элемента в БД

`bool` **`dba_exists`** `(string key, int handle)`

Возвращает `true`, если элемент с именем `key` имеется в БД `handle`, или `false` в противном случае или при ошибке.

См. также: `dba_fetch()`, `dba_delete()`, `dba_insert()`, и `dba_replace()`.

dba_firstkey. Возвращение имени первого элемента БД

`string` **`dba_firstkey`** `(int handle)`

См. также: `Dba_nextkey()`

dba_nextkey. Возвращение имени следующего элемента БД

`string` **`dba_nextkey`** `(int handle)`

См. также: `dba_firstkey()`

dba_fetch. Получение значения указанного элемента

`string` **`dba_fetch`** `(string key, int handle)`

См. также: `dba_exists()`, `dba_delete()`, `dba_insert()`, и `dba_replace()`.

dba_insert. Добавление элемента в БД

`bool` **`dba_insert`** `(string key, string value, int handle)`

Если элемент с именем `key` уже присутствует в БД, операция не выполняется. Возвращает `true`, или `false` при ошибке.

См. также: `dba_exists()`, `dba_delete()`, `dba_fetch()`, `dba_replace()`

dba_replace. Замена или добавление элемента

`bool` **`dba_replace`** `(string key, string value, int handle)`

Возвращает `true`, или `false` при ошибке.

См. также: `dba_exists()`, `dba_delete()`, `dba_fetch()`, и `dba_insert()`.

dba_delete. Удаление элемента

`string` **`dba_delete`** `(string key, int handle)`

Возвращает `true`, или `false` при ошибке.

См. также: `dba_exists()`, `dba_fetch()`, `dba_insert()`, и `dba_replace()`.

dba_optimize. Оптимизация БД

`bool` **`dba_optimize`** `(int handle)`

Возвращает `true`, или `false` при ошибке.

См. также: `dba_sync()`

dba_sync. Синхронизация изменений в БД

`bool` **`dba_sync`** `(int handle)`

Обычно физически это происходит сохранением файла на диск. Если эту операцию не выполнять изменения БД будут потеряны.

Возвращает `true`, или `false` при ошибке.

См. также: `dba_optimize()`

filePro

Функции позволяют читать данные БД filePro. См. также http://www.fileproplus.com/.

filepro. Чтение и проверка map файла

`bool` **`filepro`** `(string directory)`

Map файл содержит информацию о БД и ее структуре. Так как никакой блокировки не выполняется, постарайтесь не модифицировать БД, открывая ее в PHP.

filepro_fieldcount. Число полей в БД

```
int filepro_fieldcount(void);
```

См. также: `filepro()`.

filepro_rowcount. Число записей в БД filePro

```
int filepro_rowcount(void);
```

См. также: `filepro()`.

filepro_retrieve. Извлечение данных

```
string filepro_retrieve (int row_number, int field_number)
```

Возвращает данные поля, указанного номером `field_number` для записи `row_number`.

filepro_fieldname. Получение имени поля

```
string filepro_fieldname (int field_number)
```

filepro_fieldtype. Определение типа поля

```
string filepro_fieldtype (int field_number)
```

filepro_fieldwidth. Определение размера поля

```
int filepro_fieldwidth (int field_number)
```

Ovrimos SQL

Ovrimos SQL Server – это клиент-серверная, реляционная БД с поддержкой быстрых транзакций и веб-возможностями. http://www.ovrimos.com/. Для поддержки ovrimos в PHP необходимо скомпилировать его с параметром конфигурирования '--with-ovrimos'. Также необходимо установить библиотеку sqlcli, входящую в комплект поставки Ovrimos SQL Server.

```
<?php
$conn = ovrimos_connect("server.com", "8001", "admin", "passwrd");
if ($conn != 0) {
    echo ("Connection ok!");
      // select from a system table
    $res = ovrimos_exec ($conn,
            "select table_id, table_name from sys.tables");
    if ($res != 0) {
```

```
        echo "Statement ok!";
        ovrimos_result_all ($res);
        ovrimos_free_result ($res);
    }
    ovrimos_close($conn);
}
?>
```

ovrimos_connect. Подключение к серверу и выбор БД

`int` **`ovrimos_connect`** `(string host, string db, string user, string password)`

Возвращает дескриптор подключения, или `false` при ошибке. В аргументе `host` указывается имя сервера или его IP адрес, в `db` указывается либо имя БД, либо номер порта подключения. Также указываются имя пользователя и его пароль.

ovrimos_close. Закрытие указанного подключения

`void` **`ovrimos_close`** `(int connection)`

Все незавершенные транзакции отменяются.

ovrimos_close_all. Закрытие всех подключений

`void` **`ovrimos_close_all`** `(void)`

Все незавершенные транзакции отменяются.

ovrimos_cursor. Получение типа курсора

`int` **`ovrimos_cursor`** `(int result_id)`

Полезно при выполнении позиционированных обновлениях или удалениях.

ovrimos_longreadlen. Установка числа возвращаемых байт для длинных полей

`int` **`ovrimos_longreadlen`** `(int result_id, int length)`

По умолчанию для полей типа long varchar и long varbinary возвращается 0 байтов, но можно установить произвольное значение в аргументе `length`.

ovrimos_prepare. Подготовка запроса

`int` **`ovrimos_prepare`** `(int connection_id, string query)`

Возвращает `false` при ошибке; или дескриптор запроса, используемый для ео последующего исполнения функцией `ovrimos_execute()`.

```
<?php
$conn=ovrimos_connect ("db_host", "8001", "admin", "password");
```

```
if ($conn!=0) {
    echo "Connection ok!";
    $res=ovrimos_prepare ($conn, "select table_id, table_name
                                  from sys.tables where table_id=1");
    if ($res != 0) {
        echo "Prepare ok!";
        if (ovrimos_execute ($res)) {
            echo "Execute ok!\n";
            ovrimos_result_all ($res);
        } else {
            echo "Execute not ok!";
        }
        ovrimos_free_result ($res);
    } else {
        echo "Prepare not ok!\n";
    }
    ovrimos_close ($conn);
}
?>
```

ovrimos_execute. Выполнение подготовленного запроса

`int` **`ovrimos_execute`** `(int result_id [, array parameters_array])`

Запрос должен быть предварительно подготовлен функцией `ovrimos_prepare()`. Возвращает `true`, или `false` при ошибке. Если запрос имеет параметры (обозначенные знаком вопроса), их значения передаются в массиве `parameters_array`.

ovrimos_exec. Выполнение запроса

`int` **`ovrimos_exec`** `(int connection_id, string query)`

Возвращает дескриптор возвращенного набора записей, или `false` при ошибке. Запрос не должен содержать параметров.

ovrimos_num_rows. Получение числа измененных записей в БД

`int` **`ovrimos_num_rows`** `(int result_id)`

Функция возвращает число записей, которые были изменены запросом.

```
<?php
$conn = ovrimos_connect ("db_host", "8001", "admin", "password");
if ($conn != 0) {
    echo "Connection ok!";
    $res = ovrimos_exec ($conn, "update test set i=5");
    if ($res != 0) {
        echo "Statement ok!";
```

```
        echo ovrimos_num_rows ($res)." Записей изменено \n";
        ovrimos_free_result ($res);
    }
    ovrimos_close ($conn);
}
?>
```

ovrimos_num_fields. Получение числа полей возвращенных запросом

```
int ovrimos_num_fields (int result_id)
```

ovrimos_fetch_into. Занесение записи в массив

```
int ovrimos_fetch_into (int result_id, array result_array [, string how [, int rownumber]])
```

Функция сохраняет поля записи в массиве `result_array`, который должен передаваться по ссылке. Какая запись будет сохранена указывается в аргументе `how`, который может принимать одно из значений: 'Next' (по ум.), 'Prev', 'First', 'Last', 'Absolute' (для этого случая требуется указать номер записи в аргументе `rownumber`). Регистр не учитывается. Возвращает `true`, или `false` при ошибке.

```
<?php
$conn=ovrimos_connect ("neptune", "8001", "admin", "password");
if ($conn!=0) {
    echo "Connection ok!";
    $res=ovrimos_exec ($conn,"select t_id, t_name from tbl");
    if ($res != 0) {
        echo "Statement ok!";
        if (ovrimos_fetch_into ($res, &$row)) {
            list ($table_id, $table_name) = $row;
            echo "t_id=".$table_id.", t_name=".$table_name."\n";
            if (ovrimos_fetch_into ($res, &$row)) {
                list ($table_id, $table_name) = $row;
                echo "t_id=".$table_id.", t_name=".$table_name."\n";
            } else {
                echo "Next: error\n";
            }
        } else {
            echo "First: error\n";
        }
        ovrimos_free_result ($res);
    }
    ovrimos_close ($conn);
}
?>
```

ovrimos_fetch_row. Получение записи

`int` **`ovrimos_fetch_row`** `(int result_id [, int how [, int row_number]])`

Выбирает запись из возвращенных запросом, для того, чтобы ее данные можно было получить функцией `ovrimos_result()`. Возвращает `true`, или `false` при ошибке.

```
<?php
$conn = ovrimos_connect ("remote.host", "8001", "admin", "password");
if ($conn != 0) {
    echo "Connection ok!";
    $res=ovrimos_exec ($conn, "select t_id, t_name from tbl");
    if ($res != 0) {
        echo "Statement ok!";
        if (ovrimos_fetch_row ($res, "First")) {
            $table_id = ovrimos_result ($res, 1);
            $table_name = ovrimos_result ($res, 2);
            echo "t_id=".$table_id.", t_name=".$table_name."\n";
            if (ovrimos_fetch_row ($res, "Next")) {
                $table_id = ovrimos_result ($res, "t_id");
                $t_name = ovrimos_result ($res, "t_name");
                echo "t_id=".$table_id.", t_name=".$t_name."\n";
            } else {
                echo "Next: error\n";
            }
        } else {
            echo "First: error\n";
        }
        ovrimos_free_result ($res);
    }
    ovrimos_close ($conn);
}
?>
```

ovrimos_result. Получение поля записи

`int` **`ovrimos_result`** `(int result_id, mixed field)`

Возвращает поле записи, возвращенной запросом. Поле может быть указано его именем (или порядковым номером, начиная с 1) в аргументе `field`. Запись должна быть предварительно выбрана функцией `ovrimos_fetch_row()`.

ovrimos_result_all. Распечатка результата запроса в таблице HTML

`int` **`ovrimos_result_all`** `(int result_id [, string format])`

Возвращает `true`, или `false` при ошибке.

```
<?php
$conn = ovrimos_connect ("db_host", "8001", "admin", "password");
if ($conn != 0) {
    echo "Connection ok!";
    $res = ovrimos_prepare ($conn, "SELECT table_id, table_name
                            FROM sys.tables WHERE table_id = 7");
    if ($res != 0) {
        echo "Prepare ok!";
        if (ovrimos_execute ($res, array(3))) {
            ovrimos_result_all ($res);
        } else {
            echo "Execute not ok!";
        }
        ovrimos_free_result ($res);
    } else {
        echo "Prepare not ok!\n";
    }
    ovrimos_close ($conn);
}
?>
```

Далее расширенная версия приведенного выше скрипта, выводящая мета информацию.

Example 2. Ovrimos_result_all with meta-information

```
<?php
$conn = ovrimos_connect ("db_host", "8001", "admin", "password");
if ($conn != 0) {
    echo "Connection ok!";
    $res = ovrimos_exec ($conn, "select table_id, table_name
                                from sys.tables where table_id = 1")
    if ($res != 0) {
        echo "Statement ok! cursor=".ovrimos_cursor ($res)."\n";
        $colnb = ovrimos_num_fields ($res);
        echo "Полей = ".$colnb."\n";
        for ($i=1; $i<=$colnb; $i++) {
            $name = ovrimos_field_name ($res, $i);
            $type = ovrimos_field_type ($res, $i);
            $len = ovrimos_field_len ($res, $i);
            echo "Поле $i Имя: $name Тип: $type размер=".$len."\n";
        }
        ovrimos_result_all ($res);
        ovrimos_free_result ($res);
    }
    ovrimos_close ($conn);
}
?>
```

ovrimos_field_num. Получение порядкового номера поля

```
int ovrimos_field_num (int result_id, string field_name)
```

ovrimos_field_name. Получение имени поля

```
int ovrimos_field_name (int result_id, int field_number)
```

Поле указывается его порядковым (начиная с 1) номером `field_number`.

ovrimos_field_type. Получение типа поля

```
int ovrimos_field_type (int result_id, int field_number)
```

ovrimos_field_len. Получение размера поля

```
int ovrimos_field_len (int result_id, int field_number)
```

ovrimos_free_result. Уничтожение набора возвращенных записей

```
int ovrimos_free_result (int result_id)
```

ovrimos_commit. Завершение транзакции

```
int ovrimos_commit (int connection_id)
```

ovrimos_rollback. Отмена транзакции

```
int ovrimos_rollback (int connection_id)
```