Справочник по Perl:DBI для mSQL/mySQL

Установка

Для использования интерфейсов mSQL и MySQL к модулям DataBase Dependent/DataBase Independent (DBI/DBD) или MsqlPerl и MysqlPerl необходимо иметь следующие компоненты:

Perl 5

В вашей системе должны быть установлены работающие экземпляры Perl 5. Ко времени написания этой книги последний релиз Perl имел номер 5.005_02. Следует пользоваться по меньшей мере Perl 5.004, поскольку в более ранних версиях были ошибки, связанные с защитой. Более подробные сведения о Perl, включая адреса для загрузки, можно найти по адресу: *http://www.perl.com.*

DBI

Независимую от базы данных часть модуля DBI/DBD можно загрузить из Comprehensive Perl Archive Network (CPAN). На момент написания книги последней версией был DBI-0.90. Он находится на *http://www.perl.com/CPAN/authors/id/TIMB/DBI/DBI-l.06.tar.gz.*

Data::ShowTable

Модуль Data::ShowTable упрощает отображение больших объемов данных. Это требуется для модулей Msql-MysqL Последняя версия — Data-ShowTable-З.З, ее можно найти по адресу: *http:// www.perl.com/CPAN/authors/id/AKSTE/Data-ShowTable-3.3.tar.gz.*

mSQL и/или MySQL

В главе 3 *«Установка»* подробно рассказано о том, как получить и установить серверы баз данных mSQL и MySQL.

компилятор С и сопутствующие средства

Для модулей MsqlPerl и MysqlPerl необходим ANSI-совместимый компилятор С, а также обычные сопутствующие средства, такие как *make, Id* и т. д. Вам должно хватить тех средств, с помощью которых вы собрали Perl. Если у вас нет этих инструментов, компилятор GNU C и все необходимые поддерживающие программы можно бесплатно получить на *ftp://ftp.gnu.org/pub/gnu/.*

В настоящее время модули Msql-Mysql поддерживает Йохен Видман (Jochen Wiedmann), чье ID в CPAN выглядит как JWIED. Поэтому текущие модули Msql-Mysql всегда можно найти на *http://www.perl.com/ authors/id/JWIED.* На момент написания книги текущей версией была *Msql-Mysql-modules-1.2017.tar.gz.*

После загрузки пакета разархивируйте его:

```
tar xvzf Msql-Mysql-modules-1.2017.tar.gz cd Msql-Mysql-modules-1.2017
```

В каталоге дистрибутива есть файл *INSTALL,* содержащий несколько советов по установке. Сначала нужно выполнить файл *Make file.PL:*

```
perl Makefile.PL
```

Эта команда сначала спрашивает, желаете ли вы установить модули для mSQL, MySQL или те и другие. Можно установить модули для любого установленного вами сервера баз данных.

После проверки системы программа запрашивает местоположение установки mSQL. Это каталог, содержащий подкаталоги *lib* и *include,* в которых расположены библиотеки и включаемые файлы mSQL. По умолчанию этим каталогом является */usr/local/Hughes,* но обязательно проверьте это, поскольку во многих системах используется */usr/local* или даже */usr/local/Minerva.*

Затем сценарий установки запрашивает путь к MySQL. Как и в случае mSQL, это каталог, содержащий надлежащие подкаталоги *lib* и *include,* по умолчанию - */usr/local.* Это расположение верно для большинства установок, но следует обязательно проверить, не используются ли другие каталоги.

После установки сценарий создает соответствующие make-файлы и завершается. Затем нужно запустить *make* и скомпилировать файлы.

```
make
```

Если вы правильно установили Perl, mSQL и/или MySQL, то команда make должна пройти без ошибок. После завершения ее работы будут созданы все модули, и единственное, что останется - это протестировать и установить их.

```
make test
```

Во время выполнения по экрану пробегает ряд отладочных имен, за каждым из которых должно следовать . . . ok. И наконец, нужно установить модули.

```
make install
```

У вас должно быть разрешение записи в каталог установки Perl. Кроме того, необходимо иметь разрешение на запись в системный каталог для программ (обычно */usr/local/bin* или */usr/bin),* чтобы установить поставляемые с модулем вспомогательные программы *pmsql, pmysql* и *dbimon.*

DBI.pm API

DBI API является стандартным API баз данных в Perl. Поэтому, хотя MsqPerl и MysqlPerl могут быть более распространены в унаследованных программах, новые программы следует писать с использованием DBI.

use

```
use DBI;
```

Следует объявлять во всех программах Perl, использующих модуль DBI.

DBI::available_drivers

```
@available_drivers = DBI->available_drivers;
@available_drivers = DBI->available_drivers($quiet);
```

DBI::available_drivers возвращает список имеющихся драйверов DBD. Функция выполняет это, осуществляя поиск модулей DBD в дистрибуции Perl. Если в аргументе не передано значение true, то при обнаружении двух одноименных модулей DBD выводится предупреждение. В текущем дистрибутиве Msql-Mysql драйвер для mSQL называется 'mSQL', а драйвер для MySQL - 'mysq'.

Пример

```
use DBI;
my @drivers = DBI->available_drivers;
print "Доступны следующие драйверы:\n" . join("\n",@drivers) . "\nНо нас  интересуют только
        mSQL и mysql. :)\n";
```

DBI::bind_col

```
$result = $statement_handle->bind_col($col_num, \$c61_variable, \%unused);
```

DBI::bind_col связывает колонку команды SELECT с переменной Perl. При всяком чтении или изменении колонки изменяется значение соответствующей переменной. Первым аргументом является номер колонки в команде, при этом колонки нумеруются с 1. Второй аргумент - ссылка на переменную Perl, которая должна быть привязана к колонке. Необязательный третий аргумент ссылается на хэш атрибутов. В DBD: :mysql и DBD: :mSQL он не используется. При невозможности в силу каких-то причин сделать привязку функция возвращает неопределенное значение undef.

Пример

```
use DBI;
my  $db = DBI->Gonnect(¹DBI:mSQL:mydata',undefғundef);
```

```
my  $query = "SELECT name, date FROM myothertable";
my  $myothertable_output = $db->prepare($query);
my ($name, $date);
$nyothertable_output->bind_col(1,\$name, undef);
$myothertable_output->bind_col(2A$date, undef);
# Теперь $name и $date привязаны к соответствующим полям выходных данных,
$myothertable_output->execute;
while ($myothertable_output->fetch) {
    # Каждый раз $name и $date автоматически изменяются,
    print "Имя: $name Дата: $date\n";
}
```

DBI::bind_columns

```
$result = $statement_handle->bind_columns(\%unused, @list_of_refs_to_vars);
```

DBI::bind_columns привязывает весь список скалярных ссылок к значениям соответствующих полей в выдаче. Первый аргумент функции - ссылка на хэш атрибутов, как в DBI::bind_col . DBD::mSQL и DBD::mysql не используют этот аргумент. Последующие аргументы должны быть ссылками на скаляры. Скаляры можно с таким же успехом группировать в структуру \($var1, $var2) . Ссылок на скаляры должно быть ровно столько, сколько полей в выходных результатах, иначе выполнение программы будет прекращено.

Пример

```
use DBI;
my $db = DBI->connect('DBI:mSQL:mydata', undef, undef);
my .Squery - "SELECT name, date FROM myothertable";'
my $myothertable_output = $db->prepare($query);
my ($name,  $date);
$myothertable_output->bind_colufnns(undef,  \($name,  $date));
# $name и $date теперь привязаны к соответствующим полям в выдаче.
$myothertable_output->execute;
while ($myothertable_output->fetch) {
    # $name и $date каждый раз автоматически изменяются,
    print "Имя: $пате Дата: $date\n";
}
```

DBI::bind_param

```
$result = $statement_handle->bind_param($param_number,  $bind_value);
$result = $statement_handle->bind_param($param_number, $bind_value,  $bind_type);
$result = $statement_handle->bind_param($param_numberj $bind_value, \%bind_type);
```

DBI: :bind_param подставляет в команды действительные значения вместо меток-заполнителей *?' (см. DBI: : prepare). Первый аргумент — номер метки-заполнителя в команде, нумерация начинается с 1 (слева направо). Второй аргумент - подставляемое значение. Необязательный третий аргумент задает тип подставляемого значения. Это может быть скаляр или ссылка на хэш вида { TYPE => &DBI: :SQL_TYPE } , где 'SQL_TYPE' - тип параметра. На момент написания этой книги DBI поддерживал SQL-типы (недокументированные) SQL_CHAR, SQL_NUMERIC, SQL_DECIMAL, SQL_INTEGER, SQL_SMALLINT, SQL_FLOAT, SQL_REAL, SQL_DOUBLE и SQL_VARCHAR.. Соответствие их фактическим типам, используемым DBD::mSQL и DBD::Mysql, не документировано. Тем не менее в таблице 21-1 приведен список соответствия на данный момент. Если подстановка невозможна, функция возвращает undef.

Таблица1. *Соответствие типов SQL*

DBI	MSQL	MySQL
SQL_CHAR	CHAR TYPE	FIELD_TYPE_CHAR
	IDENT_TYPE	FIELD_TYPE_DATE

	NULL_TYPE	FIELD TYPE_DATETIME
	DATE_TYPE	FIELD TYPE_NULL
	MONEY_TYPE	FIELD__TYPE_TIMESTAMP
	TIME_TYPE	FIELD TYPE_TIME
	IDX_TYPE	
	SYSVAR_TYPE	
	ANY_TYPE	
SQL_NUMERIC		FIELD_TYPE_LONG
		FIELD_TYPE _LONGLONG
		FIELD_TYPE_SHORT
SQL_DECIMAL		FIELD_TYPE_DECIMAL
SQL_INTEGER	INT_TYPE	FIELD_TYPE_INT24
SQL_ SMALLINT	UINT_TYPE	FIELD_TYPE_INT24
SQL_FLOAT		FIELD_TYPE_FLOAT
SQL_REAL	REAL TYPE	FIELD_TYPE_DOUBLE
	LAST_REAL_TYPE	
SQL_DOUBLE		FIELD_TYPE_DOUBLE
SQL_VARCHAR	TEXT_TYPE	FIELD_TYPE_TINY_BLOB
		FIELD_TYPE_MEDIUM_BLOB
		FIELD_TYPE_BLOB
		FIELD_TYPE_LONG_BLOB
		FIELD_TYPE_VAR_STRING
		FIELD_TYPE_STRING

Пример

```
use DBI;
my $db = DBI->connect('DBD:msql:mydata', 'me', 'mypass');
my Sstatement = $db->prepare(
"SELECT name, date FROM myothertable WHERE name like ? OR name like ?");
$statement->bind_param(1,'J%', 'SQL_CHAR');
$statement->bind_param(2,'%oe%', { TYPE => &DBI::SQL_CHAR });
# Теперь команда будет такой:
# SELECT name, date FROM myothertable WHERE name like 'J%' or name like %oe%
```

DBI::connect

```
$db = DBI->connect($data_source, $username, $password);
$db = DBI->connect($data_source, Susername, $password, \%attributes);
```

DBI::connect требует по крайней мере три аргумента и необязательный четвертый. Через возвращаемый описатель выполняются все операции с сервером базы данных. Первый аргумент является источником данных. Список имеющихся источников можно получить с помощью DBI::data_sources . Для mSQL и MySQL формат источника данных 'DBI:mSQL:$database:$hostname:$port' и 'DBI:mysql:$database:$hostname:$port' соответственно. Можно опустить :$port при соединении через стандартный порт. Аналогично можно опустить : $hostname; $port при соединении с сервером на локальном узле с помощью сокета Unix. Имя базы данных указывать обязательно.

Второй и третий аргументы - имя пользователя и пароль для подключения к базе данных. Для mSQL оба аргумента должны иметь значение 'undef'. Если они заданы как 'undef' при работе с MySQL, то у пользователя, запустившего программу, должны быть права доступа к требуемым базам данных.

Последний аргумент необязателен и является ссылкой на ассоциативный массив. Данный хэш позволяет определить некоторые атрибуты соединения. В настоящее время поддерживаются только атрибуты PrintError, RaiseError и AutoCommit. Для сброса им нужно придать значение 0, для установки - какое-либо истинное значение. По умолчанию PrintError и AutoCommit включены, а RaiseError - сброшен. Поскольку в данное время ни mSQL, ни MySQL не поддерживают транзакции, атрибут AutoCommit должен быть установлен (более подробно см. Атрибуты).

При неудаче соединения возвращается неопределенное значение undef, и в $DBI: errstr помещается ошибка.

Пример

```
use DBI;
my $db1 = DBI->connect('DBI:mSQL:mydata',undef,undef);
# Теперь $db1 представляет соединение с базой данных 'mydata'на локальном
# сервере mSQL.
my $db2=DBI-> connect ('DBI:mysql:mydata:myserver.com','me','mypassword');
# Теперь $db2 представляет соединение с базой данных 'mydata1 сервера MySQL
# 'myserver.com' через порт по умолчанию.
# При соединении использовались имя пользователя 'me' и пароль 'mypassword'.
my $db3 =. DBI->connect('DBI:mSQL:mydata',undef,undef,{RaiseError => 1});
# Теперь $db3 - такое же соединение, как $db1, за исключением того, что
# установлен атрибут 'RaiseError'.
```

DBI::data_sources

```
@data_sources = DBI->data_sources($dbd_driver);
```

DBI::data_sources принимает в качестве аргумента имя модуля DBD и возвращает все имеющиеся для этого драйверы базы данных в формате, пригодном к использованию в качестве источника данных функцией DBI:::connect . Программа заканчивает свое выполнение с ошибкой, если задано неверное имя драйвера DBD. В текущих версиях модулей Msql-Mysql драйвер для mSQL называется 'mSQL', а для MySQL-'mysql'.

Пример

```
use DBI;
my @msql_data_sources = DBI->data_sources('mSOL');
my @mysql_data_sources - DBI~>data_sources('mysql');
# Должны быть установлены DBD::mSQL и DBD::mysql, иначе
# выполнение программы прекратится.
print "mSQL databases:\n" . join("\n",@msql_data_sources) , "\n\n";
print "MySQL databases:\n" . join("\n"I@mysql_data_sources) . "\n\n";
```

DBI::do

```
$rows_affected    =      $db->do($statement);
$rows_affected    =      $db->do($statement, \%unused);
$rows_affected    =      $db->do($statement, \%unused, @bind_values);
```

DBI::do непосредственно выполняет SQL-команду, не являющуюся командой SELECT, и возвращает число измененных строк. Этот способ быстрее, чем пара DBI::prepare/DBI::execute, требующая два вызова функции. Первый аргумент - сама команда SQL. Второй аргумент не используется в DBD::mSQL и DBD::mysql, но для других модулей DBD может содержать ссылку на хэш атрибутов. Последний аргумент - массив значений для подстановки в команду вместо меток-заместителей '?'. Подстановка происходит слева направо. Дополнительно DBI::do автоматически заключит подставляемые строковые значения в кавычки.

Пример

```
use DBI;
my $db = DBI->connect('DBI:mSGL:mydata',undef,undef);
my $rows_affected =
        $db->do("UPDATE mytable SET name-'Joe' WHERE name-'Bob'");
print "$rows_affected Joe заменены на Bob's\n";
my $rows_affected2 =
       $db->do("INSERT INTO mytable (name) VALUES (?)", {},'("Sheldon's Cycle"});
# После заключения в кавычки и подстановки
# серверу базы данных посылается команда
#INSERT INTO mytable (name) VALUES ('Sheldon's Cycle')
```

DBI::disconnect

```
$result = $db->disconnect;
```

DBI::disconnect отсоединяет описатель базы данных от сервера баз данных. Для mSQL и MySQL в этом обычно нет необходимости, поскольку эти базы данных не поддерживают транзакций, и неожиданное отсоединение не причинит вреда. Однако от баз данных, поддерживающих транзакции, нужно отсоединяться явным образом. Поэтому, чтобы получить переносимую программу, нужно всегда отсоединяться от базы данных перед выходом. При ошибке во время отсоединения возвращается ненулевое значение, и в $DBI::errstr устанавливается ошибка.

Пример

```
use DBI;
my $db1 = DBI->connect('DBI:mSQL:mydata", undef, undef);
my $db2 = DBI->connect('DBI:mSQL:mydata2',undef,undef);
$db1->disconnect;
# Соединение 'mydata' разорвано. Соединение с 'mydata2'
# продолжает действовать.
```

DBI::dump_results

```
$neat_rows = DBI::dump_results($statenent_handle);
$neat_rows = DBI:dump_results($statement_handle, $niaxlen);
$neat_rows = DBI::dump_results($'statement_handle, $maxlen, $line_sep);
$neat_rows - DBI::dump_results($statenient_handle, $maxlen, $line_sep,$field_sep);
$neat_rows = DBI::dump_results($statement_handle, $maxlen, $line_sep, $field_sep, $file_handle);
```

DBI:::dump_results выводит содержание описателя команды в удобном и упорядоченном виде, применяя к каждой строке DBI::neat_string. Функцию удобно использовать для быстрой проверки результатов запроса во время разработки программы. Единственный обязательный аргумент - описатель команды. Второй аргумент, если имеется, задает максимальный размер полей в таблице, по умолчанию равный 35. Третий аргумент задает строку, используемую для разграничения строк данных, по умолчанию - \n. Четвертый аргумент задает строку, используемую для разделения значений полей в строке, по умолчанию используется запятая. Последний аргумент задает ссылку на глобальный описатель файла, в который выводятся результаты. По умолчанию это STDOUT. Если невозможно прочесть описатель команды, возвращается значение undef.

Пример

```
use DBI;
my $db = DBI->connect( 'DBI:mSQL:mydata', undef, undef);
my $query = "SELECT name, date FROM myothertable";
my $myothertable_output = $db->prepare($query);
  $myothertable_output->execute;
print DBI::dump_results($myothertable_output);
# Вывести результаты в аккуратной таблице.
open (MYOTHERTABLE, "myothertable");
print DBI::dump_results($myothertable_output, undef, undef, undef, \*MYOTHERTABLE);
# Снова вывести результаты в файл 'myothertable'.
```

$DBI::err

```
$error_code = $handle->err;
```

$DBI::err возвращает код последней по времени ошибки DBI. Код ошибки соответствует сообщению об ошибке, возвращаемому функцией $DBI::errstr . Переменная $DBI::err выполняет ту же задачу. Эта функция применима с описателями как баз данных, так и команд.

Пример

```
use DBI;
my $db = DBI->connect('DBI:mysql:mydata','webuser','super_secret_squirrel');
# Этот запрос имеет синтаксическую ошибку...
my $output = $db->prepare('SLECT * from mydata');
$output->execute;
if (not $output) {
    print "Error $DBI:err:  $DBI:errstr\n";
}
```

$DBI::errstr

```
$error = $handle->errstr;
```

Эта функция возвращает сообщение о последней происшедшей ошибке DBI. Значение сохраняется до возникновения новой ошибки, когда оно будет заменено. Если во время данного сеанса ошибок не было, функция возвращает undef. Переменная $DBI::errstr выполняет ту же задачу. Эта функция применима с описателями как баз данных, так и команд.

Пример

```
Use DBI;
my $db = DBI->connect('DBI:mysql:mydata','webuser','super_secret_squirrel');
my $error = $db->errstr;
warn ("Вот последняя ошибка DBI: $error");
```

DBI::execute

```
$rows_affected = $statement_handle->execute;
$rows_affected = $statement_handle->execute(@bind_values);
```

DBI::execute выполняет SQL-команду, содержащуюся в описателе команды. Для запроса, не являющегося SELECT, функция возвращает число измененных строк. Функция возвращает -1, если число строк неизвестно. Для запроса типа SELECT при успехе возвращается истинное значение. Если заданы аргументы, они используются для подстановки имеющихся в команде меток-заместителей (см. DBI:::prepare).

Пример

```
use DBI;
my $db = DBI->connect('DBI:mSQL:mydata',undef,undef);
my $statement_handle = $db->prepare("SELECT * FROM mytable");
my $statementjiandle2 = $db->prepare("SELECT name, date FROM myothertable
       WHERE name like ?");
$statement_handle->execute;
# Выполнена первая команда.К значениям можно
# обращаться через описатель.
$statement_handle->execute("J%");
# Выполнена вторая команда следующего содержания:
# SELECT name, date FROM myothertable WHERE name like 'J%'
```

DBI::fetchall_arrayref

```
$ref_of_array_of_arrays = $statement_handle->fetchall_arrayref;
```

DBI::fetchall_arrayref возвращает все оставшиеся данные в описателе команды в виде ссылки на массив. Каждая строка массива - ссылка на другой массив, в котором содержатся данные этой строки. Если в описателе команды нет данных, функция возвращает неопределенное значение undef. Если с этим описателем команды уже выполнялись функции DBI: :fetchrow_ *, то DBI: :fetchall_arrayref возвращает все данные, оставшиеся после последнего обращения к DBI: : fetchrow_ *.

Пример

```
use DBI;
my $db = DBI->connect('DBI:mSQL:mydata',undef,undef);
my $query = "SELECT name,  date FROM myothertable";
my $output = $db->prepare($query);
$output->execute;
my $data = $output->fetchall_arrayref;
# Теперь $data является ссылкой на массив массивов.  Каждый элемент
# 'главного'  массива сам является ссылкой на массив,  содержащий строку данных.
print "Четвертой строкой данных в таблице является: " , $data->[3][1] .  "\n";
# Элемент 3  'главного' массива является массивом,  содержащим четвертую
# строку данных.
# Элемент 1 этого массива является датой.
```

DBI::fetchrow_array

```
@row_of_data = $statement_handle->fetchrow;
```

DBI::fetchrow возвращает очередную строку данных из описателя команды, созданного DBI: :execute . Каждое последующее обращение к DBI::fetchrow возвращает очередную строку данных. Когда данных больше нет, функция возвращает неопределенное значение undef. Порядок элементов в результирующем массиве определяется исходным запросом. Если запрос имел вид SELECT * FROM . . . , то элементы следуют в том порядке, в котором они были определены в таблице.

Пример

```
use DBI;
my $db = DBI->connect('DBI:mSQL:mydata',undef,undef);
my $query = "SELECT name, date FROM myothertable WHERE name LIKE 'Bob%'";
my $myothertable_output = $db->prepare($query).;
$myothertable_output->execute;
my ($name, $date);
 # Это первая строка из $myothertable_output.
($name, $date) = $myothertable_output->fetchrow_array;
# Это следующая строка...
($name, $date) = $myothertable_output->.fetchrow_array;
# И еще одна...
my @name_and_date = $myothertable_output->fetchrow_array;
 # и т.д...
```

DBI::fetchrow_arrayref, DBI::fetch

```
$array_reference = $statement_handle->fetchrow_arrayref;
$array_reference = $statement_handle->fetch;
```

DBI::fetchrow_arrayref и ее псевдоним DBI:::fetch работают точно так же, как DBI::fetchrow_array , но возвращают не фактический массив, а ссылку на него.

Пример

```
use DBI;
my $db = DBI->connect('DBI:mSQL:mydata',undef,undef);
my $query = "SELECT name, date FROM myothertable WHERE name LIKE 'Bob%'";
my $myothertable_output - $db->prepare($query);
$myothertable_output->execute;
my $name1 = $myothertable_output->fetch->[0]
# Это поле 'name' из первой строки данных,
my $date2 = $myothertable_output->fetch->[1]
# Это поле 'date' из второй строки данных,
```

```
my ($name3, $date3) = @{$myothertable_output->fetch};
# Это целиком третья строка данных.
$myothertable_output->fetch
# возвращает ссылку на массив. Можно 'преобразовать' ее в действительный
# массив, используя конструкцию @{}.
```

DBI::fetchrow_hashref

```
$hash_reference = $statement_handle->fetchrow_hashref;
```

DBI::fetchrow_hashref работает так же, как DBI::fetchrow_arrayref , но возвращает ссылку на ассоциативный, а не на обычный массив. Ключами хэша являются имена полей, а значениями - значения в этой строке данных.

Пример

```
use DBI;
my $db = DBI->connect('DBI:mSQL:mydata', undef, undef);
my $query = "SELECT * FROM mytable";
my $mytable_output = $db->prepare($query);
$mytable_output->execute;
my %row1 = $mytable_ouput->fetchrow_hashref;
my @field_names = keys %row1;
# @field_names содержит-теперь имена всех полей в запросе.
# Это делается только один раз. Во всех следующих строках будут те же поля.
my @row1 = values %row1;
```

DBI::finish

```
$result = $statement_handle->finish;
```

DBI::finish освобождает все данные в описателе команды, чтобы можно было уничтожить описатель или снова подготовить его. Некоторым серверам баз данных это необходимо для освобождения соответствующих ресурсов. DBD::mSQL и DBD::mysql не нуждаются в этой функции, но для переносимости кода следует использовать ее по окончании работы с описателем команды. Функция возвращает неопределенное значение undef, если описатель не удается освободить.

Пример

```
use DBI;
my $db - DBI->connect('DBI:mysql:mydata', 'me','mypassword');
my $query = "SELECT * FROM mytable";
my $mytable_output - $db->prepare($query);
$mytable_output->execute;
$mytable_output->finish;
# Теперь можно переназначить $mytable_output или подготовить для него
# другую команду SQL.
```

DBI::func

```
$handle->func(@func_arguments, $func_name);
@dbs = $db->func("$hostname", '_ListDBs');
@dbs = $db->func("$hostname:Sport", '_ListDBs');
@tables = $db->func('_ListTables');
$result = $drh~>func( $database, '_CreateDB' );
$result = $drh->func( $database, '_DropDB' );
```

DBI::func вызывает специализированные непереносимые функции, включенные в различные драйверы DBD. Она используется с описателем базы данных или описателем команды, в зависимости от назначения специализированной функции. По возможности следует использовать равносильную переносимую функцию. При использовании специализированной функции сначала передаются ее аргументы как скаляр, а затем - имя

функции. DBD::mSQL и DBD::mysql реализуют следующие функции:

_ListDBs

Функция _ListDBs принимает имя узла и необязательный номер порта и возвращает список имеющихся у сервера баз данных. Лучше использовать переносимую функцию DBI::data_sources .

_Listlat)les

Функция _ListTables возвращает список таблиц, имеющихся в текущей базе данных.

_CreateDB

Функция _CreateDB принимает в качестве аргумента имя базы данных и пытается создать эту базу данных на сервере. Для работы с этой функцией необходимо иметь право создания баз данных. Функция возвращает -1 в случае неудачи и 0 в случае успеха.

_DropDB

Функция _DropDB принимает в качестве аргумента имя базы данных и пытается удалить с сервера эту базу данных. Данная функция не выводит пользователю сообщений и при успешном выполнении удаляет базу данных навсегда. Для работы с этой функцией необходимо иметь право удаления баз данных. Функция возвращает -1 в случае неудачи и 0 в случае успеха.

Пример

```
use DBI;
my $db = DBI->connect('DBI:mysql:mydata','me1, 'mypassword');
my @tables = $db->func('_ListTables');
# @tables содержит теперь список таблиц в 'mydata'.
```

DBI::neat

$neat_string.= DBI::neat($string); $neat_string - DBI: :neat(.$string, $maxlen);

DBI::neat принимает в качестве аргументов строку и необязательную длину. Затем строка форматируется для аккуратного вывода. Вся строка заключается в одиночные кавычки. Непечатаемые символы заменяются точками. Если задан аргумент длины, все символы после максимальной длины удаляются, а строка заканчивается тремя точками (...). Если длина не указана, по умолчанию используется значение 400.

Пример

```
use DBI;
my $string = "Это очень, очень длинная строка, в которой много чего написано.";
my $neat_string = DBI::neat($string,14);
# Теперь $neat_string такая: 'Это очень, оче..."
```

DBI::neat_list

$neat_string = DBI;:neat_list(\@listref, Smaxlen);
$neat_string = DBI::neat_list(\@listref, $maxlen, $fi§ld_seperator);

DBI: :neat_list принимает три аргумента и возвращает аккуратно отформатированную строку, пригодную для вывода. Первый аргумент содержит ссылку на список выводимых значений. Второй аргумент -максимальная длина каждого поля. Последний аргумент - строка, используемая для разделения полей. Для каждого элемента списка вызывается DBI:: neat с использованием заданной максимальной длины. В результирующих строках для разделения полей используется последний аргумент. Если последний аргумент не задан, в качестве разделителя применяется запятая.

Пример

```
use DBI;
my @list = ('Bob',   'Joe',   'Frank');
```

```
my $neat_string = DBI::neat_list(\@list, 3);
# Теперь $neat_string такая: 'Bob', 'Joe', 'Fra...'
```

DBI::prepare

```
$statement_handle = $db->prepare($statement); $statement_handle = $db-
>prepare($statement, \%unused);
```

DBI::prepare принимает в качестве аргумента SQL-команду, которую некоторые модули баз данных переводят во внутреннюю компилированную форму, исполняемую быстрее при вызове DBI:: execute . Эти модули DBD (не DBD::mSQL или DBD::mysql) принимают также ссылку на хэш необязательных атрибутов. Серверы mSQL и MySQL в настоящее время не реализуют концепцию подготовки команд, поэтому DBI:: prepa re просто запоминает команду. По желанию вместо значений данных в команду можно вставить любое количество символов '?'. Эти символы известны как метки-заместители (placeholders). Функция DBI: ;bincLparam осуществляет подстановку действительных значений вместо меток-заместителей. Если по какой-либо причине команду нельзя подготовить, функция возвращает undef.

Пример

```
use DBI;
my $db = DBI->connect('DBI:mysql:mydata', 'me','mypassword');
my $statement_handle = $db->prepare('SELECT * FROM mytable');
# Теперь эта команда готова к выполнению.
My $statement_handle = $db->prepare(
'SELECT name, date FROM myothertable WHERE name like ?');
# Эта команда будет готова к выполнению после подстановки
# с использованием функции DBI::bind_param.
```

DBI::quote

```
$quoted_string = $db->quote($string);
```

DBI::quote принимает строку для использования в качестве запроса SQL и возвращает ее копию с правильно расставленными для ввода в запрос кавычками, в том числе расставляя корректные кавычки по концам строки.

Пример

```
use DBI;
my $db1 = DBI->connect('DBI:mSQL:mydata',undef,undef);
my $db2 = DBI->connect('DBI:mysql:myotherdata', 'me','mypassword');
my Sstring = "Sheldon's Cycle";
my $qs1 = $db1->quote($string);
# $qs1: 'Sheldon\'s Cycle' (включая наружные кавычки)
my $qs2 = $db2->quote($string);
# $qs2 : 'Sheldon's Cycle' (включая наружные кавычки) .
# Теперь обе строки годятся для использования в командах для своих
# соответствующих серверов баз данных.
```

DBI::rows

```
$number_of_rows = $statement_handle->rows;
```

DBI: : rows возвращает число строк данных, содержащихся в описателе команды. Для DBD::mSQL и DBD::mysql эта функция дает точное число для всех команд, включая SELECT. Для многих других драйверов, которые не хранят в памяти сразу все результаты, эта функция надежно работает только для команд, не являющихся SELECT. Это следует учитывать при написании переносимых программ. Функция возвращает -1, если по какой-либо причине число строк неизвестно. Переменная $DBI: ; rows выполняет ту же задачу.

Пример

```
use DBI;
my $db = DBI->connect('DBI:mSQL:mydata',undef,undef);
my $query = "SELECT name,  date FROM myothertable WHERE name='Bob'";
my $myothertable_output = $db->prepare($query);
$myothertable_output->execute;
my $rows = $myothertable_output->rows;
print "В таблице 'myothertable' есть $rows строк 'Вов'\n";
```

DBI::state

```
$sql_error = $handle->state;
```

DBI:::state возвращает код ошибки SQL SQLSTATE последней по времени ошибки DBI. В данное время DBD::mSQL и DBD;:mysql сообщают 'S1000' для всех ошибок. Эта функция доступна для описателей баз данных и команд. Переменная $DBI::state выполняет ту же задачу.

Пример

```
use DBI;
my $db = DBI->connect('DBI:mysql:mydata', 'webuser', 'super_secret_squirrel');
my $sql_error = $db->state;
warn("Вот последняя по времени ошибка DBI SQL: $sql_error");
```

DBI::trace

```
DBI->trace($trace_level)
DBI->trace($trace_level, $trace_file)
$handle->trace($trace_level);
$handle->trace($trace_level, $trace_file);
```

DBI::trace используется в основном для отладки. Если уровень трассировки установлен равным 2, выводится полная отладочная информация. Установка уровня 0 отключает трассировку. DBI->trace осуществляет трассировку для всех описателей, а $handle->trace - только для данного описателя - базы данных или команды. При наличии DBI->trace или $handle->trace второго аргумента отладочная информация выводится в указанный файл. Также трассировку можно включить, установив значение переменной окружения DBI_TRACE. Если переменная окружения установлена равной числу (в настоящее время 0 или 2), включается трассировка всех описателей на этом уровне. При другом значении переменной уровень трассировки устанавливается равным 2, а само значение используется в качестве имени файла для вывода отладочной информации.

Пример

```
use DBI;
my $db1 = DBI->connect('DBI:mysql:mydata', 'webuser','super_secret_squirrel');
my $db2 = DBI->connect('DBI:mSQL:myotherdata',undef,undef);
DBI->trace(2);
# Включена трассировка для всех описателей на уровне 2.
$db2->trace(0);
# Отключена трассировка для $db2, но продолжает действовать для $db1
$db1->trace(2,'DBI.trace');
# Теперь включена трассировка для всех описателей на уровне 2, выдача
# посылается в файл 'DBI.trace'.
```

DBI::commit
DBI::rollback
DBI::ping

```
$result = $db->commit;
$result = $db->rollback;
```

```
$result = $db->ping;
```

DBI::commit и DBI::rollback полезны только при работе с серверами, поддерживающими транзакции. При работе с DBD::mSQL и DBD::mysql они не оказывают никакого эффекта. DBD: :ping пытается проверить, запущен ли сервер базы данных. В DBD::mSQL и DBD: :mysql она не реализована.

Атрибуты

```
$db->{AutoCommit}
$handle->{ChopBlanks}
$handle->{CompatMode}
$handle->{InactiveDestroy}
$handle->{LongReadLen}
$handle->{LongTruncOk}
$handle->{PrintError}
$handle->{RaiseError}
$handle->{Warn}
$statement_handle->{CursorName}
$statement_handle->{insertid} (только MySQL)
$statement_handle->{is_blob} (только MySQL)
$statement_handle->{is_key} (только MySQL)
$statement_handle->{is_not_null}
$statement_handle->{is_nufri}
$statement_handle->{is_pri_key} (только MySQL и mSQL 1.x
$statement_handle->{length}
$statement_handle->{max_length} (только MySQL)
$statement_handle->{NAME}
$statement_handle->{NULLABLE}
$statement_handle->{NUM_OF_FIELDS}
$statement_handle->{NUM_OF_PARAMS}
$statement_handle->{table}
$statement_handle->{type}
```

В DBI.pm API определено несколько атрибутов, которые можно читать и устанавливать в любой момент. Присвоение значения атрибуту может определенным образом изменить поведение текущего соединения. Присвоение любого отличного от нуля значения атрибуту устанавливает его. Присвоение значения 0 атрибуту сбрасывает его. Некоторые значения определены только для конкретных баз данных и непереносимы. Ниже следуют атрибуты, применимые как к описателям баз данных, так и к командам.

$db->{AutoCommit}

Этот атрибут оказывает влияние на поведение серверов баз данных, поддерживающих транзакции. Для mSQL и MySQL он всегда должен быть установлен (значение по умолчанию). Попытка изменить его прерывает выполнение программы.

$handle->{ChopBlanks}

При установке этого атрибута отсекаются все ведущие и замыкающие пробелы в данных, возвращаемых запросом (например, при вызове DBI::fetch row).. Все производные от данного описатели наследуют значение этого атрибута. Значение по умолчанию - «сброшен».

$handle->{InactiveDestroy}

Назначение этого атрибута- сохранить процесс при ветвлении *(fork),* чтобы дочерний процесс мог пользоваться родительским описателем. Его следует установить в родительском или дочернем процессе, но не в обоих. Значение по умолчанию - «сброшен».

$handle-> {PrintError}

При установке этого атрибута выводятся все предупредительные сообщения. При сброшенном

атрибуте доступ к ошибкам осуществляется только через $DBI::errstr. Все производные от данного описатели наследуют значение этого атрибута. Значение по умолчанию -«установлен».

$handle->{RaiseError)

При установке этого атрибута все ошибки возбуждают в программе исключительные ситуации, прерывая ее выполнение, если не определен обработчик '_DIE_'. Все описатели, производные от этого, наследуют значение этого атрибута. Значение по умолчанию -«сброшен».

$handle->{Warn}

При установке этого атрибута выводятся предупредительные сообщения о неудачных приемах программирования (особенно пережитках Perl 4). Сброс этого атрибута отключает предупреждения DBI, что допустимо только при полной уверенности в своем мастерстве. Все производные от данного описатели (например, описатель команды, происходящий от описателя базы данных) наследуют значение этого атрибута. Значение по умолчанию - «установлен».

$statement_handle->{insertid}

Непереносимый атрибут, определенный только для DBD::mysql. Он возвращает из таблицы текущее значение поля auto_increment (если таковое имеется). Если поле auto_increment не существует, атрибут возвращает undef.

$statement_handle->{is_blob}

Это непереносимый атрибут, определенный только для DBD::mysql. Атрибут возвращает ссылку на массив булевых значений, указывающих для каждого из содержащихся в описателе команды полей, имеет ли оно тип BLOB. Для описателя команды, который был создан не выражением SELECT, $statement_handle->{is_blob} возвращает undef.

$statement_handle->{is_key}

Непереносимый атрибут, определенный только для DBD::mysql. Он возвращает ссылку на массив булевых значений, указывающих для каждого из содержащихся в описателе команды полей, определено ли оно как KEY. Для описателя команды, который был создан не выражением SELECT, $statement_handle->{is_key} возвращает undef.

$statement_handle->{is_not_null}

Это непереносимый атрибут, определенный только для DBD::mSQL и DBD::mysql. Он возвращает ссылку на массив булевых значений, указывающих для каждого из содержащихся в описателе команды полей, определено ли оно как 'NOT NULL' . Для описателя команды, который был создан не выражением SELECT, данный атрибут возвращает undef. Того же результата можно достичь в переносимом виде, используя $statement_handle->{NULLABLE} .

$statement_handle->{is_num}

Это непереносимый атрибут, определенный только для DBD::mSQL и DBD::mysql. Атрибут возвращает ссылку на массив булевых значений, указывающих для каждого из содержащихся в описателе команды полей, имеет ли оно числовой тип. Для описателя команды, созданного не выражением SELECT, $statement_handle->{is_num} возвращает undef.

$statement_handle->{is_pri_key}

Это непереносимый атрибут, определенный только для DBD::mSQL и DBD::mysql. При использовании с DBD::mSQL он оказывает влияние только для серверов mSQL1.x, поскольку mSQL2.x не использует первичные ключи. Атрибут возвращает ссылку на массив булевых значений, указывающих для каждого из содержащихся в описателе команды полей, является ли оно первичным ключом.

Для описателя команды, созданного не выражением SELECT, данный атрибут возвращает undef.

$statenent_handle->{length}

Непереносимый атрибут, определенный только для DBD::mSQL и DBD::mysql. Этот атрибут возвращает ссылку на список максимально допустимых размеров полей, содержащихся в описателе команды. Для описателя команды, который был создан не выражением SELECT, $statement_handle->{length} возвращает undef.

$statement_handle->{.max_length}

Это непереносимый атрибут, определенный только для DBD::mysql. Атрибут возвращает ссылку на список фактических максимальных размеров полей, содержащихся в описателе команды. Для описателя команды, который был создан не выражением SELECT, данный атрибут возвращает undef.

$statement_handle->{NAME}

Атрибут возвращает ссылку на список имен полей, содержащихся в описателе команды. Для описателя команды, который был создан не выражением SELECT, $statement_handle->{NAME} возвращает undef.

$statement_handle->{NULLABLE}

Этот атрибут возвращает ссылку на массив булевых значений, указывающих для каждого из содержащихся в описателе команды полей, может ли оно иметь значение NULL. Поле, определенное как 'NOT NULL', даст в списке значение 0. Остальные поля дадут значение 1. Для описателя команды, созданного не выражением SELECT, атрибут возвращает undef.

$statement_handle->{NUM_OF_FIELDS}

Атрибут возвращает число колонок данных, содержащихся в описателе команды. Для описателя команды, который был создан не выражением SELECT, $statement_handle->{NUM_OF_FIELDS} возвращает 0

$statementJandle->{NUM_OF_PARAMS}

Этот атрибут возвращает число меток-заместителей в описателе команды. Метки-заместители обозначаются в команде символом '?'. Для подстановки вместо меток-заместителей надлежащих значений используется функция DBI:: bind_values .

$statement_handle->{table}

Это непереносимый атрибут, определенный только для DBD: rmSQL и DBD::mysql. Атрибут возвращает ссылку на список имен таблиц, к которым осуществлялся доступ в запросе. Полезно использовать для SELECT с соединением нескольких таблиц.

$statement_handle->{type}

Непереносимый атрибут, определенный только для DBD::mSQL и DBD::mysql. Он возвращает ссылку на список типов полей, содержащихся в описателе команды. Для описателя команды, созданного не выражением SELECT, $statement_handle->{max_length} возвращает undef. Значениями списка являются целые числа, соответствующие перечислению в заголовочном файле C mysql_com.h из дистрибутива MySQL. Сейчас способа доступа к именам этих типов из DBI не существует. Но доступ к типам возможен через функцию &Mysql::FIELD_TYPE_* в Mysql.pm. В DBD::mysql имеется также недокументированный атрибут $statement_handle->{format_type_name}, идентичный $statement_handle->{type}, за исключением того, что вместо целых чисел возвращает SQL-названия типов. *Следует подчеркнуть, что это недокументированный атрибут, и автор* DBD::mysql *высказал свое намерение убрать его, как только в DBI будет реализована такая же функция.*

$statement_handle->{CursorName}
$handle->{LongReadLen}
$handle->{LongTruncQk}
$handle->{CompatMode}

Все эти атрибуты не поддерживаются в DBD::mSQL и DBD::mysql. Присвоение им значений ничего не даст, а чтение возвратит 0 или undef. Исключение составляет атрибут $statement handle->{CursorName} . В настоящее время любое обращение к нему «убьет» программу.

Пример

```
use DBI;
my $db = DBI->connect('mysqL:mydata','me','mypassword');
$db->{RAISE_ERROR} - 1;
# Теперь любая ошибка DBI/DBD убьет программу.
my $statement_handle = $db->prepare('SELECT•* FROM mytable');
$statement_handle->execute;
my @fields = @{$statement_handle->{NAME}};
# @fields содержит теперь список с именами всех полей в 'mytable'.
```