СИСТЕМА
УПРАВЛЕНИЯ
БАЗАМИ
ДАННЫХ

ЛИНТЕР®

ЛИНТЕР БАСТИОН

ЛИНТЕР СТАНДАРТ

Perl-интерфейсы

НАУЧНО-ПРОИЗВОДСТВЕННОЕ ПРЕДПРИЯТИЕ

РЕЛЭКС

Товарные знаки

РЕЛЭКС™, ЛИНТЕР® являются товарными знаками, принадлежащими АО НПП «Реляционные экспертные системы» (далее по тексту – компания РЕЛЭКС). Прочие названия и обозначения продуктов в документе являются товарными знаками их производителей, продавцов или разработчиков.

Интеллектуальная собственность

О документе

Материал, содержащийся в данном документе, прошел доскональную проверку, но компания РЕЛЭКС не гарантирует, что документ не содержит ошибок и пропусков, поэтому оставляет за собой право в любое время вносить в документ исправления и изменения, пересматривать и обновлять содержащуюся в нем информацию.

Контактные данные

394006, Россия, г. Воронеж, ул. Бахметьева, 2Б.

Тел./факс: (473) 2-711-711, 2-778-333.

e-mail: market@relex.ru.

Техническая поддержка

С целью повышения качества программного продукта ЛИНТЕР и предоставляемых услуг в компании РЕЛЭКС действует автоматизированная система учёта и обработки пользовательских рекламаций. Обо всех обнаруженных недостатках и ошибках в программном продукте и/или документации на него просим сообщать нам в раздел Поддержка на сайте ЛИНТЕР.

Содержание

Предисловие

Назначение документа

Документ содержит описание Perl-интерфейсов с СУБД ЛИНТЕР.

Документ предназначен для СУБД ЛИНТЕР СТАНДАРТ 6.0 сборка 20.1, далее по тексту СУБД ЛИНТЕР.

Для кого предназначен документ

Документ предназначен для программистов, разрабатывающих приложения на языке программирования Perl с использованием СУБД ЛИНТЕР.

Необходимые предварительные знания

Для работы необходимо:

- знать основы реляционных баз данных и языка баз данных SQL;
- знать спецификации DBX;
- знать язык программирования Perl;
- уметь работать в соответствующей операционной системе на уровне простого пользователя.

Дополнительные документы

- СУБД ЛИНТЕР. Сетевые средства
- СУБД ЛИНТЕР. Системные таблицы и представления
- СУБД ЛИНТЕР. Интерфейс нижнего уровня
- СУБД ЛИНТЕР. Справочник по SQL
- СУБД ЛИНТЕР. Архитектура СУБД
- СУБД ЛИНТЕР. Справочник кодов завершения

LinPerl-интерфейс

Назначение

Perl – это система разработки скриптов, включающая в себя CGI-интерфейс (Common Gateway Interface – стандартный шлюзовой интерфейс, предназначенный для создания серверных приложений HTTP), интерпретатор языка и набор функций для доступа к базам данных (БД) и различным объектам Web-сервера. Perl является удобным средством разработки приложений WWW и интерфейсов к БД в Интернет.

Условия применения

Для использования LinPerl-интерфейса СУБД ЛИНТЕР на хост-компьютере должен быть установлен интерпретатор языка программирования Perl. Если интерпретатор Perl не установлен, то его исходные тексты и инструкцию по установке можно найти на сайте https://www.perl.org.

Для выполнения Perl-программы на компьютере должен быть установлен интерфейс DBI версии 1.43 или выше.

Исходные тексты интерфейса DBI можно найти по ссылке https://metacpan.org/release/DBI.

Характеристика программы

В системе Perl СУБД ЛИНТЕР всегда рассматривается как удаленный SQL-сервер, то есть для доступа к БД создается сетевое соединение. Благодаря этому возможно открывать из одного скрипта либо несколько пользовательских сессий, либо работать с различными SQL-серверами. После установки соединения с сервером можно отправлять и обрабатывать SQL-запросы. При выполнении запроса создается объект, в котором хранится результат выполнения запроса, после чего с помощью специальных функций можно получать отдельные записи запроса.

Сборка для ОС типа Windows

Перед сборкой драйвера необходимо задать в переменной окружения LINTER_HOME путь к установочному каталогу СУБД ЛИНТЕР:

```
set LINTER_HOME=<LINTER_HOME_PATH>
```

Дистрибутив LinPerl-интерфейса для ОС Windows возможно собрать двумя способами:

1) с помощью компилятора MSVC:

   в консоле разработчика Visual Studio перейти в каталог с исходными файлами библиотеки `<LINTER_HOME>\intlib\PERL` и выполнить команды:

```
perl Makefile.PL
nmake
nmake install
```

2) с помощью утилиты `dmake`:

   выполнить команды:

```
perl Makefile.PL
dmake
dmake install
```

Примечания

1. В случае отсутствия утилиты необходимо выполнить ее установку с помощью менеджера пакетов Perl:

```
ppm install dmake
```

2. Путь до утилиты `dmake` должен быть в переменной окружения PATH.

После успешного построения драйвера в текущем каталоге будет создан файл `LinPerl.pm`.

Для работы с LinPerl-интерфейсом необходимо скопировать файл `LinPerl.pm` в один из каталогов библиотек Perl. Полный список каталогов библиотек Perl можно получить с помощью команды 'perl -V' или изменить значение переменной окружения PERL5LIB:

```
set PERL5LIB=%PERL5LIB%;%LINTER_HOME%\intlib\PERL
```

Сборка для ОС типа UNIX

Для сборки дистрибутива:

1) задать в переменной окружения LINTER_HOME путь к установочному каталогу СУБД ЛИНТЕР:

```
export LINTER_HOME=<LINTER_HOME_PATH>
```

2) перейти в каталог с исходными файлами библиотеки $LINTER_HOME/perl и выполнить команды:

```
perl Makefile.PL
make -f Makefile
make -f Makefile install
```

После успешного построения драйвера в подкаталоге /bin установочного каталога СУБД ЛИНТЕР будут созданы файлы `LinPerl.pm` и `LinPerl.so`. Для работы с LinPerl-интерфейсом необходимо добавить в переменную окружения PATH путь до файлов `LinPerl.pm` и `LinPerl.so`.

Функции

Общие сведения

1) Все символьные параметры функций чувствительны к регистру символов.
2) Если длина символьного параметра функции превышает максимально допустимую в СУБД ЛИНТЕР, то значение параметра усекается до допустимой длины и функция выполняется с усеченным значением параметра.
3) Все функции возвращают результат типа INTEGER.

   Возможные коды завершения:

   - 0 (LPE_SUCCESS) - нормальное завершение, возвращаемое значение отсутствует;
   - положительное - возвращаемое функцией значение (нормальное завершение);
   - отрицательное - неудачное завершение функции.
4) Все возвращаемые значения передаются через параметр(ы) функции.
5) Причиной неудачного завершения функции может являться как ошибка Perl-модуля, так и результат обработки функции СУБД ЛИНТЕР. Если причина в СУБД ЛИНТЕР, то возвращается код завершения `LPE_LINTER_ERROR`. Все остальные отрицательные коды относятся к ошибкам Perl-модуля (приложение 1). Для получения дополнительной информации об ошибке используйте функцию `GetErrorMsg` (пункт «Получить описание кода завершения СУБД ЛИНТЕР»).

Установить соединение с СУБД

Назначение

Функция `OpenConnect` устанавливает связь между Perl-модулем и СУБД ЛИНТЕР на узле с заданным именем пользователя и паролем.

Синтаксические правила

```
OpenConnect (<пользователь>, <пароль>, <сервер>, <режим>,
 <идентификатор соединения>);
```

`<Пользователь>`

Имя пользователя БД. Регистрозависимая символьная строка длиной не более 66 символов.

`<Пароль>`

Пароль пользователя. Регистрозависимая символьная строка длиной не более 18 символов.

`<Сервер>`

Имя ЛИНТЕР-сервера, с которым необходимо установить соединение. Символьная строка длиной не более 8 символов. Если параметр не задан, предполагается локальный компьютер.

`<Режим>`

Задает режим обработки транзакций и кодовую страницу (значение 0 или логическая комбинация опций `<режим транзакции>` и `<кодовая страница>`).

<Режим транзакции>:

- TM_AUTOCOMMIT;
- TM_OPTIMISTIC;

Примечание
Режим TM_OPTIMISTIC устарел (использовать не рекомендуется).

- TM_EXCLUSIVE.

<Кодовая страница>:

- CP_1251;
- CP_KOI8;
- CP_866.

Если значение параметра равно 0, значения опций используются по умолчанию.

Возвращаемое значение

<Идентификатор соединения> с СУБД.

Пример

```
$err = OpenConnect("KENT", "ALEX", "LINTER", CP_866 |
 MODE_EXCLUSIVE, $con);
$err &&  [code for handling error]
```

См. также функции:

CloseConnect.

Получить параметры соединения с СУБД

Назначение

Получить информацию о параметре БД, с которой установлено соединение, или о параметре самого соединения с СУБД.

Синтаксические правила

```
GetConnectInfo(<идентификатор соединения>, <ассоциативный
 массив>);
```

<Идентификатор соединения>

Идентификатор установленного соединения с СУБД.

<Ассоциативный массив>

Адрес ассоциативного массива.

Возвращаемое значение

<Ассоциативный массив> параметров БД (таблица 1).

Таблица 1. Ассоциативный массив параметров БД

Ключ массива	Возвращаемое значение ключа
VERMAJOR	Старшая версия СУБД
VERMINOR	Младшая версия СУБД
VERBUILD	Номер сборки СУБД
SORTPOOLSIZE	Размер пула сортировки (в страницах по 4 Кбайт)
KERNELPOOLSIZE	Размер пула ядра СУБД (в страницах по 4 Кбайт)
FILEQUEUESIZE	Размер очереди файлов
USERQUEUESIZE	Размер очереди пользователей
TABLEQUEUESIZE	Размер очереди таблиц
COLUMNQUEUESIZE	Размер очереди столбцов
CHANNELQUEUESIZE	Размер очереди каналов
SNAPTIMEOUT	Период времени между операциями полного Snap
KILLTIMEOUT	Тайм-аут опроса существования клиента
BASENAME	Имя БД
SYSLOG	Признак активности журнала транзакций.
SYNC	Признак синхронизации ввода/вывода
LOG	Признак ведения файла-протокола
OS	Идентификатор ОС
NUMOFSORT	Количество файлов сортировки
FLAGS_CSETCHG	Кодовая страница не найдена, используется по умолчанию
FLAGS_KERNELENGLOCALE	Используется англоязычная кодовая страница.
FLAGS_EXTPASS	Пароль пользователю дан администратором, должен быть изменен
FLAGS_PASSEND	Пароль пользователя должен быть изменен
FLAGS_KERNELINVBYTEORDER	Порядок байт на сервере и клиенте совпадает/не совпадает
FLAGS_KERNELDEMOLIC	БД с демо-лицензией
FLAGS_KERNELDEMOLICEXP	Срок лицензии на демо-БД закончился/не закончился
MAXRECSIZE	Максимальный размер записи в таблице
CHARSET	Текущая кодировка БД
DEFCHARSET	Кодировка по умолчанию БД
USECHARSET	Установлена пользовательская кодировка соединения
USECHARSETNAME	Имя установленной пользовательской кодировки

Пример

```
$err = GetConnectInfo($cur, \%hash);
$build = $hash{"VERBUILD"};
$os = $hash{"OS"};
```



```
$err && [code for handling error]
```

См. также функции:

OpenConnect, GetColInfo.

Установить пользовательскую кодировку

Назначение

Установить новую кодировку во всех активных соединениях приложения с СУБД и во всех курсорах, созданных в этих соединениях.

Синтаксические правила

```
SetCodepage (<кодовая страница>);
```

```
<Кодовая страница>
```

Имя кодовой страницы.

Пример

```
$err = SetCodepage("KOI8-R");
$err &&  [code for handling error]
```

См. также функции:

OpenConnect, GetConnectInfo.

Закрыть соединение с СУБД

Назначение

Закрыть установленное с СУБД соединение.

Синтаксические правила

```
CloseConnect(<идентификатор соединения>);
```

```
<Идентификатор соединения>
```

Идентификатор ранее установленного соединения с СУБД.

Примечание
Если соединение является родителем одного или нескольких курсоров, то курсоры тоже будут закрыты.

Пример

```
$err = CloseConnect($con);
$err &&  [code for handling error]
```

См. также функции:

OpenConnect.

Открыть курсор

Назначение

Открыть курсор между PERL-модулем и СУБД ЛИНТЕР.

Синтаксические правила

```
OpenCursor(<идентификатор соединения>, <курсор>, <идентификатор
 курсора>);
```

```
<Идентификатор соединения>
```

Идентификатор родительского соединения.

```
<Имя курсора>
```

Символьная строка длиной до 18 символов.

```
<Идентификатор курсора>
```

Переменная языка Perl

Примечание
В текущей версии параметр `<имя курсора>` не поддерживается.

Возвращаемое значение

`<Идентификатор курсора>`.

Примечание
Режим транзакции и кодовая страница создаваемого курсора наследуются из родительского соединения.

Пример

```
$err = OpenCursor($con, "MY_CURSOR", $cur);
$err &&  [code for handling error]
```

См. также функции:

OpenConnect, CloseCursor.

Получить значение параметра курсора

Назначение

Получить значение параметров курсора.

Синтаксические правила

```
GetCursorOption(<идентификатор курсора>, <параметр курсора>,
 <значение параметра>);
```

<Идентификатор курсора>

Идентификатор созданного курсора.

<Параметр курсора>

Идентификатор запрашиваемого значения параметра курсора (таблица 2).

<Значение параметра>

Переменная языка Perl.

Таблица 2. Идентификаторы параметров курсора

Параметр курсора	Описание
CO_CURNAME	Имя курсора
CO_ASYNCMODE	Режим асинхронного выполнения курсора
CO_SYNCMODE	Режим синхронного выполнения курсора
CO_ASYNCDONE	Уведомление о завершении асинхронного вызова
CO_DTFORMAT	Формат по умолчанию данных типа DATE (DD.MM.YYYY HH:MI:SS.FF)
CO_COLCOUNT	Количество столбцов в ответе
CO_ROWCOUNT	Количество строк в ответе
CO_ERRROW	Количество строк в сообщении об ошибке
CO_ERRPOS	Указывать местоположение ошибки в строке
CO_TRANSMODE	Режим транзакций
CO_CURROW	Текущая строка в запросе выборки
CO_CURROWID	RowId текущей строки
CO_CONNECTID	Идентификатор соединения
CO_NODENAME	Имя ЛИНТЕР-сервера, с которым установлено соединение
CO_PRIORITY	Приоритет курсора
CO_QUERY_PRIORITY	Приоритет текущего запроса
CO_CANCEL	Отмена выполнения запроса
CO_PAUSE	Приостанов выполнения запроса
CO_CONTINUE	Возобновление выполнения запроса
CO_LAST_ROWID	Предоставление идентификатора последней записи. Должен использоваться только после INSERT-запроса
CO_FETCH_BLOBS_AS_USUAL_DATA	Загружать в строку ответа BLOB-данные (при отключенной опции загрузка BLOB-данных игнорируется). Должен использоваться только после INSERT-запроса

Возвращаемое значение

Значение запрошенного параметра в переменной <значение параметра>.

Пример

```
$err = GetCursorOption($cursor, CO_ROWCOUNT, $rows);
$err && ... to do something for error handling ...
```

См. также функции:

SetCursorOption, GetConnectInfo, GetColInfo.

Установить значение параметра курсора

Назначение

Установить значение параметра курсора.

Синтаксические правила

```
SetCursorOption(<идентификатор курсора>, <параметр курсора>,
 <значение параметра>);
```

`<Идентификатор курсора>`

Идентификатор созданного курсора.

`<Параметр курсора>`

Идентификатор параметра курсора, которому устанавливается новое значение (таблица 2).

`<Значение параметра>`

Переменная языка Perl, содержащая новое значение параметра.

Пример

```
$err = SetCursorOption($cursor, CO_ PRIORITY, 10);
$err && ... to do something for error handling ...
```

См. также функции:

GetCursorOption.

Закрыть курсор

Назначение

Закрыть ранее открытый курсор.

Синтаксические правила

```
CloseCursor(<идентификатор курсора>);
```

`<Идентификатор курсора>`

Идентификатор созданного курсора.

Пример

```
$err = CloseCursor($cur);
```



```
$err &&  [code for handling error]
```

См. также функции:

OpenCursor.

Выполнить курсорный запрос без подготовки

Назначение

Выполнить курсорный запрос без предварительной подготовки.

Синтаксические правила

```
ExecDirect(<идентификатор курсора>, <SQL-запрос>);
```

`<Идентификатор курсора>`

Идентификатор созданного курсора.

`<SQL-запрос>`

Символьная строка, содержащая текст SQL-запроса.

Примечание
Текст SQL-запроса должен завершаться символом «;».

Пример

```
$err = ExecDirect($cur, "select * from auto;");
$err && [code for handling error]
```

См. также функции:

Fetch, Prepare, BindParameter, BindParamArray, Execute.

Подготовить курсорный запрос для выполнения

Назначение

Подготовка курсорного запроса для выполнения.

Синтаксические правила

```
Prepare(<идентификатор курсора>, <SQL-запрос>);
```

`<Идентификатор курсора>`

Идентификатор созданного курсора.

`<SQL-запрос>`

Символьная строка, содержащая текст SQL-запроса с параметрами.

Примечание
Текст SQL-запроса должен завершаться символом «;».

Пример

```
$err = Prepare($cur, "select * from auto where personid = ?;");
$err &&  [code for handling error]
```

См. также функции:

BindParameter, BindParamArray, Execute, ExecDirect.

Привязать значение параметра к курсорному запросу

Назначение

Привязать значения параметра к курсорному запросу.

Синтаксические правила

```
BindParameter(<идентификатор курсора>, <параметр>, <значение
 параметра>);
```

`<Идентификатор курсора>`

Идентификатор созданного курсора.

`<Параметр>`

Номер параметра в претранслированном (подготовленном с помощью функции Prepare) SQL-запросе в данном курсоре. Нумерация параметров начинается с `1`.

`<Значение параметра>`

Значение, которое должно быть присвоено указанному параметру.

Примечание
Если ранее уже была сделана привязка параметра, то она будет отменена.

Пример

```
$err = BindParameter($cur, 1, $my_param);
$err &&  [code for handling error]
```

См. также функции:

Prepare, Execute, BindParamArray.

Привязать массив значений параметров к курсорному запросу

Назначение

Привязать массив значений параметров к курсорному запросу.

Синтаксические правила

```
BindParamArray(<идентификатор курсора>, <массив значений>[,
 <начальный номер>, <конечный номер>]);
```

`<Идентификатор курсора>`

Идентификатор созданного курсора.

`<Массив значений>`

Число элементов в массиве параметров должно быть больше или равно количеству параметров в подготовленном запросе.

`<Начальный номер>`

Номер параметра в претранслированном (подготовленном с помощью функции Prepare) SQL-запросе в данном курсоре, начиная с которого должна выполняться привязка параметров.

`<Конечный номер>`

Номер параметра в претранслированном (подготовленном с помощью функции Prepare) SQL-запросе в данном курсоре, в котором должна закончиться привязка параметров.

Нумерация параметров начинается с `1`.

Если аргументы `<начальный номер>` или `<конечный номер>` не заданы, по умолчанию `<начальный номер>` берется равным 1, а `<конечный номер>` – последнему номеру параметра в подготовленном SQL-запросе.

Примечание
Если ранее уже была сделана привязка параметра, то она будет отменена.

Пример

```
$err = BindParamArray($cur, 1, 3, \@my_param);
$err &&  [code for handling error]
```

См. также функции:

Prepare, Execute, BindParameter.

Выполнить подготовленный курсорный запрос

Назначение

Выполнение подготовленного курсорного запроса.

Синтаксические правила

```
Execute(<идентификатор курсора>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором следует выполнить последний подготовленный SQL-запрос.

Пример

```
$err = Execute($cur);
$err &&  [code for handling error]
```

См. также функции:

ExecDirect, Prepare, BindParameter, BindParamArray, Fetch.

Перейти в заданную строку ответа курсорного запроса

Назначение

Перемещение в заданную строку ответа курсорного запроса.

Синтаксические правила

```
Fetch(<идентификатор курсора>, <направление перемещения>, <номер
 строки>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором выполнен запрос выборки данных.

`<Направление перемещения>`

Задает новое местоположение указателя текущей строки ответа.

Возможные значения параметра:

- `FETCH_FIRST` – перемещение на первую строку;
- `FETCH_LAST` – перемещение на последнюю строку;
- `FETCH_NEXT` – перемещение на следующую строку;
- `FETCH_PREV` – перемещение на предыдущую строку;
- `FETCH_ABSNUM` – перемещение на заданную строку.

`<Номер строки>`

Целочисленное положительное значение, задающее порядковый номер строки, в которую надо переместиться в текущей выборке. Параметр используется только с параметром `FETCH_ABSNUM`.

Нумерация строк начинается с `1`.

Пример

```
$err = Fetch($cur, FETCH_NEXT, 0);
$err = Fetch($cur, FETCH_ABSNUM, 100);
$err &&  [code for handling error]
```

См. также функции:

ExecDirect, Execute.

Привязать переменную к столбцу

Назначение

Привязать переменную к столбцу ответа курсорного запроса.

Синтаксические правила

```
BindColumn(<идентификатор курсора>, <номер столбца>,
 <переменная>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором выполнен запрос выборки данных.

`<Номер столбца>`

Целочисленное положительное значение, задающее номер столбца в строке выборки, к которому надо привязать переменную. Нумерация столбцов начинается с `1`.

`<Переменная>`

Задает переменную, в которую будет помещаться текущее значение выбранного столбца при перемещении по строкам ответа курсорного запроса.

Возвращаемое значение

После выполнения курсорного запроса значение заданного столбца текущей строки ответа будет находиться в привязанной переменной.

Если значение столбца NULL, возвращается NULL-значение.

Если тип данных столбца BLOB и включена настройка `CO_FETCH_BLOBS_AS_USUAL_DATA`, то возвращается содержимое BLOB-столбца, иначе – пустая строка.

Если тип столбца BOOLEAN, то возвращается строка TRUE или FALSE.

Если тип столбца EXTFILE, то возвращается ошибка.

Пример

```
$err = BindColumn($cur, 1, $make);
$err &&  [code for handling error]
```

См. также функции:

GetDataColumn, Fetch, UnbindColumn.

Отменить привязку переменной к столбцу

Назначение

Отменить ранее произведенную привязку переменной к столбцу ответа курсорного запроса.

Синтаксические правила

```
UnbindColumn(<идентификатор курсора>, <номер столбца>);
```

<Идентификатор курсора>

Идентификатор курсора, в котором выполнен запрос выборки данных.

<Номер столбца>

Целочисленное положительное значение, задающее номер столбца в строке выборки, для которого надо отменить привязку переменной. Нумерация столбцов начинается с 1.

Пример

```
$err = UnbindColumn($cur, 1);
$err &&  [code for handling error]
```

См. также функции:

BindColumn.

Получить значение столбца

Назначение

Получить в переменной значение заданного столбца текущей строки ответа курсорного запроса.

Синтаксические правила

```
GetDataColumn(<идентификатор курсора>, <номер столбца>,
 <переменная>);
```

<Идентификатор курсора>

Идентификатор курсора, в котором выполнен запрос выборки данных.

<Номер столбца>

Целочисленное положительное значение, задающее номер столбца в строке выборки, значение которого необходимо получить. Нумерация столбцов начинается с 1.

<Переменная>

Задает переменную, в которую будет помещено текущее значение выбранного столбца строки ответа курсорного запроса.

Возвращаемое значение

Значение столбца в заданной переменной.

Если значение столбца NULL, значение переменной также NULL.

Если тип данных столбца BLOB и включена настройка CO_FETCH_BLOBS_AS_USUAL_DATA, то возвращается содержимое BLOB-столбца, иначе – пустая строка.

Если тип столбца BOOLEAN, то возвращается строка TRUE или FALSE.

Если тип столбца EXTFILE, то возвращается ошибка.

Примечание
Для получения значения столбца типа BLOB рекомендуется использовать функцию `BLOBGetData`.

Пример

```
$err = GetDataColumn($cur, 1, $make);
$err &&  [code for handling error]
```

См. также функции:

ExecDirect, Execute, Fetch, BLOBGetData, GetDataRow.

Получить массив значений строки ответа

Назначение

Получить массив значений текущей строки ответа курсорного запроса.

Синтаксические правила

```
GetDataRow(<идентификатор курсора>, <массив>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором выполнен запрос выборки данных.

`<Массив>`

Адрес массива для записи значений строки ответа.

Возвращаемое значение

Массив значений столбцов текущей строки ответа курсорного запроса. Фактическое количество помещенных в массив значений зависит от размера выделенного массива

Пример

```
$err = ExecDirect($cur, "select * from auto;");
$err && ... to do something for error handling ...
$err = GetDataRow($cur, \@res);
$err && ... to do something for error handling ...
$make=$res[0];
$model=$res[1];
```

См. также функции:

ExecDirect, Execute, Fetch, GetDataColumn, GetDataArray, GetDataRowS.

Получить массивы значений нескольких строк ответа

Назначение

Получить массивы значений нескольких строк ответа курсорного запроса.

Синтаксические правила

```
GetDataRowS | GetM (<идентификатор курсора>, <массив>, <количество
 строк>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором выполнен запрос выборки данных.

`<Массив>`

Адрес массива для записи значений строк ответа.

`<Количество строк>`

Целое неотрицательное значение. Отсчет начинается с `1`.

Возвращаемое значение

Двумерный массив значений заданного количества строк ответа курсорного запроса.

Отсчет строк ответа начинается с текущей строки курсорного запроса.

Если в курсорном запросе нет нужного количества строк ответа, возвращается сколько есть.

Примечание
С функциональной точки зрения функции `GetDataRowS` и `GetM` идентичны. Отличие в том, что при выполнении функции `GetM` СУБД ЛИНТЕР использует пакетную загрузку данных в массив, что при большом количестве запрашиваемых строк может дать выигрыш по времени.

Пример

```
$err = ExecDirect($cur, "select * from auto;");
$err && ... to do something for error handling ...
$err = GetDataRowS($cur, \@res, 5);
$err && ... to do something for error handling ...
$make1=$res[0][0]; # марка машины в первой строке (1-й столбец)
$model1=$res[0][1] # модель машины в первой строке (2-й столбец)
…
$make3=$res[2][0]; # марка машины в третьей строке (1-й столбец)
$model3=$res[2][1] # модель машины в третьей строке (2-й столбец)
```

См. также функции:

ExecDirect, Execute, Fetch, GetDataColumn, GetDataRow, GetDataArray.

Получить ассоциативный массив значений строки ответа

Назначение

Получить ассоциативный массив значений текущей строки ответа курсорного запроса.

Синтаксические правила

```
GetDataArray(<идентификатор курсора>, <массив>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором выполнен запрос выборки данных.

`<Массив>`

Адрес ассоциативного массива для записи значений строк ответа.

Возвращаемое значение

Ассоциированный массив значений текущей строки ответа курсорного запроса.

Пример

```
$err = ExecDirect($cur, "select * from auto;");
$err && ... to do something for error handling ...
$err = GetDataArray($cur, \%res);
$err && ... to do something for error handling ...
$make=$res{"MAKE"};
$model=$res{"MODEL"};
```

См. также функции:

ExecDirect, Execute, Fetch, GetDataColumn, GetDataRow.

Проверить значение столбца на NULL-значение

Назначение

Проверить значение заданного столбца текущей строки ответа курсорного запроса на NULL-значение.

Синтаксические правила

```
TestNull(<идентификатор курсора>, <номер столбца>, <результат
 проверки>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором выполнен запрос выборки данных.

`<Номер столбца>`

Целочисленное положительное значение, задающее номер столбца в текущей строке выборки, значение которого необходимо проверить. Нумерация столбцов начинается с `1`.

`<Результат проверки>`

Переменная, в которую будет помещен результат проверки на NULL-значение.

Возвращаемое значение

Результат проверки в переменной:

- `1` – значение столбца NULL;

- 0 – значение столбца не NULL.

Пример

```
$err = TestNull($cur, 1, $is_null);
$err &&  [code for handling error]
if ($is_null != 0) ...
```

См. также функции:

GetDataColumn, BLOBGetData, GetDataRow, GetDataArray.

Добавить нетипизированные данные к первому BLOB-столбцу

Назначение

Добавить порцию данных к первому найденному BLOB-столбцу (если такой имеется) в текущей строке курсорного запроса.

Синтаксические правила

```
BLOBAppend(<идентификатор курсора>, <порция данных>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором выполнен запрос выборки данных.

`<Порция данных>`

Адрес буфера, содержащего добавляемую порцию данных.

Возвращаемое значение

Порция данных добавляется в конец существующих BLOB-данных.

По умолчанию типу добавляемых BLOB-данных присваивается значение 0 (устанавливается при добавлении первой порции BLOB-данных).

Пример

```
$err = BLOBAppend($cur, $blob);
$err &&  [code for handling error]
```

См. также функции:

BLOBClear, BLOBGetSize, BLOBGetData, BLOBAdd, BLOBAppendEx.

Добавить типизированные данные к первому BLOB-столбцу

Назначение

Добавить типизированную порцию данных к первому найденному BLOB-столбцу (если такой имеется) в текущей строке курсорного запроса.

Синтаксические правила

```
BLOBAppendEx(<идентификатор курсора>, <порция данных>, <тип
 данных>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором выполнен запрос выборки данных.

`<Порция данных>`

Адрес буфера, содержащего добавляемую порцию данных.

`<Тип данных>`

Целочисленное значение, характеризующее тип BLOB-данных.

> **Примечание**
> Значения типа данных от -11 по -20 зарезервированы СУБД ЛИНТЕР для внутреннего использования.

Возвращаемое значение

Порция данных добавляется в конец существующих BLOB-данных.

Тип BLOB-данных устанавливается при добавлении только первой порции данных, при добавлении остальных порций игнорируется.

Пример

```
$err = BLOBAppendEx($cur, $blob, $blob_type);
$err &&  [code for handling error]
```

См. также функции:

BLOBAppend, BLOBClear, BLOBGetSize, BLOBGetData, BLOBAddEx.

Добавить нетипизированные данные к заданному BLOB-столбцу

Назначение

Добавить порцию данных к заданному BLOB-столбцу (если такой имеется) в текущей строке курсорного запроса.

Синтаксические правила

```
BLOBAdd(<идентификатор курсора>, <порция данных>, <номер
 столбца>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором выполнен запрос выборки данных.

`<Порция данных>`

Адрес буфера, содержащего добавляемую порцию данных.

<Номер столбца>

Целочисленное положительное значение, задающее номер столбца в текущей строке выборки, к которому необходимо добавить порцию данных. Нумерация столбцов начинается с 1

Возвращаемое значение

Порция данных добавляется в конец существующих BLOB-данных.

По умолчанию типу добавляемых BLOB-данных присваивается значение 0 (устанавливается при добавлении первой порции BLOB-данных).

Пример

```
$err = BLOBAdd($cur, $blob);
$err &&  [code for handling error]
```

См. также функции:

BLOBAppend, BLOBAddEx, BLOBPurge, BLOBFetch.

Добавить типизированные данные к заданному BLOB-столбцу

Назначение

Добавить типизированную порцию данных к заданному BLOB-столбцу (если такой имеется) в текущей строке курсорного запроса .

Тип BLOB-данных устанавливается при добавлении только первой порции данных, при добавлении остальных порций игнорируется.

Синтаксические правила

```
BLOBAddEx(<идентификатор курсора>, <порция данных>, <номер
 столбца>, <тип данных>);
```

<Идентификатор курсора>

Идентификатор курсора, в котором выполнен запрос выборки данных.

<Порция данных>

Адрес буфера, содержащего добавляемую порцию данных.

<Номер столбца>

Целочисленное положительное значение, задающее номер столбца в текущей строке выборки, к которому необходимо добавить порцию данных. Нумерация столбцов начинается с 1

<Тип данных>

Целочисленное значение, характеризующее тип BLOB-данных.

Примечание
Значения типа данных от -11 по -20 зарезервированы СУБД ЛИНТЕР для внутреннего использования.

Возвращаемое значение

Порция данных добавляется в конец существующих BLOB-данных.

Тип BLOB-данных устанавливается при добавлении только первой порции данных, при добавлении остальных порций игнорируется.

Пример

```
$err = BLOBAddEx($cur, $blob,3, $blob_type);
$err && [code for handling error]
```

См. также функции:

BLOBAppend, BLOBAppendEx, BLOBAdd, BLOBPurge, BLOBFetch.

Получить тип BLOB-данных

Назначение

Получить тип BLOB-данных заданного BLOB-столбца (если такой имеется) в текущей строке курсорного запроса.

Синтаксические правила

```
BLOBGetType(<идентификатор курсора>, <тип данных>,
 <номер столбца>);

<Идентификатор курсора>
```

Идентификатор курсора, в котором выполнен запрос выборки данных.

```
<Тип данных>
```

Целочисленная переменная языка Perl.

```
<Номер столбца>
```

Целочисленное положительное значение, задающее номер столбца в текущей строке выборки, для которого необходимо получить тип его BLOB-данных. Нумерация столбцов начинается с `1`.

Возвращаемое значение

Тип BLOB-данных в переменной `<тип данных>`.

Пример

```
$err = BLOBGetType($cur, $blob_type, $column);
$err &&  [code for handling error]
```

См. также функции:

BLOBAppendEx, BLOBAddEx, BLOBGetSize.

Удалить BLOB-данные первого BLOB-столбца

Назначение

Удалить полностью BLOB-данные первого найденного BLOB-столбца (если такой имеется) текущей строки курсорного запроса.

Синтаксические правила

```
BLOBClear(<идентификатор курсора>);
```

```
<Идентификатор курсора>
```

Идентификатор курсора, в котором выполнен запрос выборки данных.

Пример

```
$err = BLOBClear($cur);
$err &&  [code for handling error]
```

См. также функции:

BLOBAppend, BLOBGetSize, BLOBGetData, BLOBPurge.

Удалить BLOB-данные заданного BLOB-столбца

Назначение

Удалить полностью BLOB-данные заданного BLOB-столбца (если такой имеется) текущей строки курсорного запроса.

Синтаксические правила

```
BLOBPurge(<идентификатор курсора>, <номер столбца>);
```

```
<Идентификатор курсора>
```

Идентификатор курсора, в котором выполнен запрос выборки данных.

```
<Номер столбца>
```

Целочисленное положительное значение, задающее номер столбца в текущей строке выборки, у которого необходимо удалить его BLOB-данные. Нумерация столбцов начинается с 1.

Пример

```
$err = BLOBPurge($cur, $column);
$err &&  [code for handling error]
```

См. также функции:

BLOBAppend, BLOBAdd, BLOBGetSize, BLOBGetData, BLOBFetch, BLOBClear.

Получить размер BLOB-данных

Назначение

Получить размер BLOB-данных заданного BLOB-столбца текущей строки курсорного запроса.

Синтаксические правила

```
BLOBGetSize(<идентификатор курсора>, <размер> [,<номер столбца>]);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором выполнен запрос выборки данных.

`<Размер>`

Переменная языка Perl.

`<Номер столбца>`

Целочисленное положительное значение, задающее номер столбца в текущей строке выборки, для которого необходимо получить тип его BLOB-данных. Нумерация столбцов начинается с `1`. Если аргумент не задан, по умолчанию используется `1`.

Возвращаемое значение

Размер BLOB-данных (в байтах) в переменной `<размер>`.

Пример

```
$err = BLOBGetSize($cur, $blob_size);
$err &&  [code for handling error]
```

См. также функции:

BLOBAppend, BLOBAdd, BLOBClear, BLOBPurge, BLOBGetData, BLOBFetch, BLOBGetType.

Получить порцию BLOB-данных первого BLOB-столбца

Назначение

Получить порцию BLOB-данных первого найденного BLOB-столбца (если такой имеется) текущей строки курсорного запроса.

Синтаксические правила

```
BLOBGetData(<идентификатор курсора>, <начало порции>, <размер>,
 <буфер>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором выполнен запрос выборки данных.

<Начало порции>

Относительный номер байта, с которого начинается требуемая порция данных. Нумерация байтов начинается с 1.

<Размер>

Переменная языка Perl, задающая размер требуемой порции данных.

<Буфер>

Буфер для размещения порции данных.

Возвращаемое значение

Порция BLOB-данных в заданном <буфере>.

Реальная длина переданных данных в переменной <размер>.

Пример

```
$err = BLOBGetData($cur, 1, 1000, $blob);
$err &&  [code for handling error]
```

См. также функции:

BLOBAppend, BLOBClear, BLOBGetSize, GetDataColumn, BLOBFetch.

Получить порцию BLOB-данных заданного BLOB-столбца

Назначение

Получить порцию BLOB-данных заданного BLOB-столбца (если такой имеется) текущей строки курсорного запроса.

Синтаксические правила

```
BLOBFetch(<идентификатор курсора>, <начало порции>, <размер>,
 <буфер>, <номер столбца>);
```

<Идентификатор курсора>

Идентификатор курсора, в котором выполнен запрос выборки данных.

<Начало порции>

Относительный номер байта, с которого начинается требуемая порция данных. Нумерация байтов начинается с 1.

<Размер>

Переменная языка Perl, задающая размер требуемой порции данных.

<Буфер>

Буфер для размещения порции данных.

<Номер столбца>

Целочисленное положительное значение, задающее номер столбца в текущей строке выборки, из которого необходимо извлечь порцию BLOB-данных. Нумерация столбцов начинается с 1. Если аргумент не задан, по умолчанию используется 1.

Возвращаемое значение

Порция BLOB-данных в заданном <буфере>.

Реальная длина переданных данных в переменной <размер>.

Пример

```
$err = BLOBFetch($cur, 1, 1000, $blob, $column);
$err &&  [code for handling error]
```

См. также функции:

BLOBAppend, BLOBAdd, BLOBClear, BLOBPurge, BLOBGetData, GetDataColumn.

Подтверждение выполнения транзакции

Назначение

Подтвердить изменения, внесенные в БД, при выполнении текущей транзакции курсорного запроса.

Синтаксические правила

```
Commit(<идентификатор курсора>);
```

<Идентификатор курсора>

Идентификатор курсора, в котором внесены изменения в БД.

Возвращаемое значение

Все изменения данных, выполненные при обработке транзакции, фиксируются в БД.

Пример

```
$err = Commit($cur);
$err &&  [code for handling error]
```

См. также функции:

ExecDirect, Execute, Rollback.

Отмена выполнения транзакции

Назначение

Отменить изменения, внесенные в БД, при выполнении текущей транзакции курсорного запроса.

Синтаксические правила

```
Rollback(<идентификатор курсора>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором внесены изменения в БД.

Возвращаемое значение

Все изменения данных, выполненные при обработке транзакции, отменяются и в БД не сохраняются.

Пример

```
$err = Rollback($cur);
$err &&  [code for handling error]
```

См. также функции:

ExecDirect, Execute, Commit.

Получить описание атрибутов столбца

Назначение

Получить описание атрибутов заданного столбца текущей строки курсорного запроса.

Синтаксические правила

```
GetColInfo(<идентификатор курсора>, <номер столбца>, <массив>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором выполнен запрос выборки данных.

`<Номер столбца>`

Целочисленное положительное значение, задающее номер столбца в текущей строке выборки. Нумерация столбцов начинается с `1`.

`<Массив>`

Адрес массива для значений атрибутов столбца.

Возвращаемое значение

Ассоциативный массив значений атрибутов столбца (таблица 3).

Таблица 3. Ассоциативный массив значений атрибутов столбца

Ключ массива	Возвращаемое значение ключа
is_null	Столбец допускает (значение 1)/не допускает (значение 0) NULL-значение.
type	Числовое значение типа данных столбца (приложение 2).

Ключ массива	Возвращаемое значение ключа
type_name	Имя типа данных столбца (приложение 2). Имя указывается без префикса LDT_.
length	Длина данных.
precision	Точность представления (для вещественных значений).
scale	Масштаб (для вещественных значений).
name	Имя столбца.
table	Имя таблицы, которой принадлежит столбец.
user	Имя владельца таблицы.

Пример

```
$err = GetColInfo($cur, 1, \%data);
$err &&  [code for handling error]
```

См. также функции:

ExecDirect, Execute, GetConnectInfo, GetCursorOption.

Получить последний код завершения

Назначение

Получить код завершения СУБД ЛИНТЕР или операционной системы последнего курсорного запроса.

Синтаксические правила

```
GetError(<идентификатор курсора>, <код СУБД>, <код ОС>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором выполнен SQL-запрос.

`<Код СУБД>`

Переменная языка Perl.

`<Код ОС>`

Переменная языка Perl.

Возвращаемое значение

Переменная	Значение
`<код СУБД>`	Код завершения последнего выполненного SQL-запроса в данном курсоре
`<код ОС>`	Код завершения операционной системы, соответствующий последнему выполненному SQL-запросу в данном курсоре

Пример

```
$err = GetError($cur, $linerr, $syserr);
```

```
$err &&  [code for handling error]
```

См. также функции:

GetErrorMsg.

Получить описание кода завершения СУБД ЛИНТЕР

Назначение

Получение описания кода завершения СУБД ЛИНТЕР.

Синтаксические правила

```
GetErrorMsg(<идентификатор курсора>, <код СУБД>, <текст
 сообщения>);
```

`<Идентификатор курсора>`

Идентификатор курсора, в котором выполнен SQL-запрос.

`<Код СУБД>`

Числовой код завершения обработки SQL-запроса ядром СУБД ЛИНТЕР.

`<Текст сообщения>`

Переменная языка Perl.

Возвращаемое значение

Текстовое описание заданного кода завершения СУБД ЛИНТЕР в переменной `<текст сообщения>`.

Примечание
Для выполнения функции в БД должна существовать таблица `SYSTEM.ERRORS`. В противном случае возвращается пустая строка.

DBI-интерфейс

Общие сведения

DBI-интерфейс (Data Base Interface) является интерфейсом доступа к БД, написанным в форме Perl-модуля. Этот модуль определяет набор «методов» и «атрибутов», через которые осуществляется доступ к БД из Perl-программы. Конкретная реализация методов специфична для каждой БД и реализуется с помощью DBD-драйверов (Data Base Driver), разрабатываемых для каждой СУБД. Этот драйвер также имеет форму Perl-модуля.

DBI работает как переключатель между Perl-программой и DBD-драйвером.

Более подробно о DBI можно узнать на сайте.

Коды завершения DBI-интерфейса представлены в приложении 3.

Необходимые условия

Драйвер требует версии Perl не ниже 5.006 и DBI не ниже 1.43.

Установка в ОС типа UNIX

Для сборки и установки DBD-драйвера необходимо:

1) задать переменную окружения LINTER_HOME – установочный каталог СУБД ЛИНТЕР:

```
export LINTER_HOME=<LINTER_HOME_PATH>
```

2) перейти в каталог с исходными файлами библиотеки `$LINTER_HOME/perl-dbi` и выполнить команды:

```
perl Makefile.PL
make -f Makefile
make -f Makefile install
```

После успешного выполнения команд все необходимые файлы для работы драйвера будут размещены в соответствующих рабочих каталогах Perl. Получить информацию о том, какие файлы были установлены и в какие каталоги, можно в результате выполнения последней команды. В том числе узнать о списке имен каталогов, в которых Perl производит поиск библиотек, можно, выполнив команду (см. значение переменной @INC):

```
perl -V
```

Убедиться в работоспособности интерфейса можно, выполнив пример из каталога `samples/DBI` дистрибутива ЛИНТЕР.

Установка в ОС типа Windows

Для сборки установки DBD-драйвера необходимо:

1) в консоле разработчика Visual Studio перейти в каталог с исходными файлами библиотеки `<LINTER_HOME>\intlib\Perl\DBD`:

2) выполнить команды:

```
perl Makefile.PL
nmake
nmake install
```

После успешного выполнения команд все необходимые файлы для работы драйвера будут размещены в соответствующих рабочих каталогах Perl. Получить информацию о том, какие файлы были установлены и в какие каталоги, можно в результате выполнения последней команды.

В том числе узнать о списке имен каталогов, в которых Perl производит поиск библиотек, можно, выполнив команду (см. значение переменной @INC):

```
perl -V
```

Динамические атрибуты DBI

Динамические атрибуты содержат информацию о результате выполнения последнего вызванного метода. Их значения должны анализироваться или использоваться до вызова другого метода.

Атрибут	Описание
DBI::errmsg	Числовой код завершения последнего выполненного метода
DBI::errstr	Текст кода завершения последнего выполненного SQL-запроса к СУБД ЛИНТЕР
DBD::Linter::VERSION	Версия DBD-драйвера СУБД ЛИНТЕР

Методы

Создать новый объект класса Linter (install_driver)

Назначение

Метод `install_driver` позволяет создать новый объект класса Linter.

Пакет

Package Linter

Прототип

```
$drh = DBI->install_driver('Linter');
```

Возвращаемые значения

Переменная	Описание
`$drh`	Описатель драйвера (в случае нормального завершения) и undef – в случае внутренней ошибки DBI

Примечание
Метод автоматически вызывается методом `install_driver` в DBI.

Создать новое соединение (connect)

Назначение

Метод `connect` создает новое соединения с СУБД и объект класса «соединение». Вызов метода может быть произведен как через объект класса Linter, так и непосредственно через класс DBI.

Пакет

Package Linter::dr

Прототип

```
$dbh = $drh->connect($data_source,$username,$password[, \%attr]);

$dbh = DBI->connect($data_source, $username, $password [, \
%attr]);
```

Параметр	Описание
$data_source	Строка подключения вида `"dbi:Linter(database => <dbname>, mode => {autocommit | optimistic | exclusive}, codepage => {CP_1251|CP_866,CP_UTF8|KOI8|<codepage_name>}):"` где: <dbname> – имя узла локальной сети, на котором установлена СУБД ЛИНТЕР. Имя этого узла и параметры доступа к нему должны быть прописаны в файле сетевой конфигурации `nodetab` (см. документ «СУБД ЛИНТЕР. Сетевые средства»). Для доступа к локальной СУБД ЛИНТЕР по умолчанию значение <dbname> должно быть пустым. <codepage_name> – имя кодовой страницы, присутствующей в системной таблице $$$CHARSET (см. документ «СУБД ЛИНТЕР. Системные таблицы и представления»). Все параметры подключения (кроме <dbname>) можно задать через параметр %attr). **Примечания** 1. Наличие двоеточия в конце строки подключения после закрывающей скобки обязательно. 2. Режим OPTIMISTIC устарел (использовать не рекомендуется).
$username	Имя пользователя БД (регистр символов учитывается) длиной не более 66 символов. Если длина имени больше 66 символов, то она усекается
$password	Пароль пользователя БД (регистр символов учитывается) длиной не более 18 символов. Если длина пароля больше 18 символов, то она усекается
\%attr	Ссылка на список атрибутов создаваемого соединения:

Параметр	Описание
	%attributes= (mode => {autocommit | optimistic | exclusive}, codepage => {CP_866 | CP_1251 | CP_UTF8 | KOI8 | <codepage_name>}) где: <codepage_name> – имя кодовой страницы, присутствующей в системной таблице $$$CHARSET (см. документ «СУБД ЛИНТЕР. Системные таблицы и представления»). Значения по умолчанию: mode => autocommit codepage => CP_866 **Примечание** Режим OPTIMISTIC устарел (использовать не рекомендуется).

Примечание
При одновременном задании атрибутов подключения в строке подключения и через параметр %attr используются значения, заданные в параметре %attr.

Например:

```
%attr = (mode => exclusive, codepage => CP_866);
$dbh = $drh->connect("dbi:Linter(database => DEMO, mode =>
 optimistic, codepage => CP_1251)", "SYSTEM", "MANAGER8", \%attr);
```

В итоге соединение будет открыто в транзакционном режиме EXCLUSIVE с установленной кодовой страницей CP_866.

Возвращаемые значения

Переменная	Описание
$dbh:	
описатель БД	Нормальное завершение
undef	Ошибка соединения с СУБД
$DBI::errstr	Диагностическое сообщение (в случае ошибки соединения с СУБД)

Создание кэшированного соединения (connect_cached)

Назначение

Метод connect_cached выполняет соединение с ЛИНТЕР-сервером аналогично методу connect и в случае успешного соединения сохраняет параметры этого соединения в кэше для возможного повторного соединения к этому же ЛИНТЕР-серверу. Если вызов метода connect_cached выполняется повторно с теми же значениями

параметров соединения, что уже присутствует в кэше, то метод возвратит этот экземпляр соединения. Кэшированный экземпляр соединения заменяется новым при закрытии соединения с ЛИНТЕР-сервером или после неуспешного выполнения метода `ping`.

Примечание
Механизм кэширования соединений отличается от механизма постоянных соединений, осуществляемых Apache::DBI. Однако, если Apache::DBI загружен, то `connect_cached` будет использовать его.

Не рекомендуется менять атрибуты в экземпляре соединения, созданном посредством вызова `connect_cached`, так как это может негативно отразиться там, где используется то же самое соединение. После вызова `connect_cached` возвращается дескриптор соединения, у которого установленные атрибуты будут сброшены в исходные значения (например, атрибут `AutoCommit`).

Пакет

Package Linter::dr

Прототип

```
$dbh = $drh->connect_cached($data_source, $username, $password [,\
%attr]);
$dbh = DBI->connect_cached($data_source, $username, $password [,\
%attr]);
```

Параметр	Описание
$data_source	См. описание метода connect
$username	См. описание метода connect
$password	См. описание метода connect
\%attr	См. описание метода connect

Возвращаемые значения

Переменная	Описание
`$dbh`	Ссылка на дескриптор открытого соединения
`undef`	Ошибка соединения с СУБД ЛИНТЕР
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки выполнения метода)

Пример

```
$dbh = $drh->connect_cached("dbi:Linter(database => DEMO):",
 "SYSTEM", "MANAGER8");
```

Создание копии дескриптора соединения (clone)

Назначение

Метод `clone` создает дубликат (копию) дескриптора $dbh соединения с первоначальными параметрами соединения ($dsn, $user, $password), возможно, с дополнительными указаниями драйверу по обработке дублируемого соединения. Метод

также может использоваться для дублирования дескрипторов соединений, которые разъединены с СУБД в данный момент.

Пакет

Package Linter::db

Прототип

```
$new_dbh = $dbh->clone(\%attr);
```

Параметр	Описание
\%attr	См. описание метода connect

Возвращаемые значения

Переменная	Описание
`$new_dbh`	Копия соединения или undef при возникновении ошибки
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Проверка доступности ЛИНТЕР-сервера (ping)

Назначение

Метод `ping` используется для проверки доступности ЛИНТЕР-сервера через ранее установленное соединение. Причиной отказа в доступе может быть:

- неактивное состояние ядра СУБД ЛИНТЕР;
- недоступность узла, на котором установлен ЛИНТЕР-сервер.

Пакет

Package Linter::db

Прототип

```
$rc = $dbh->ping;
```

Возвращаемые значения

Переменная	Описание
`$rc`	Результат проверки доступа к ЛИНТЕР-серверу: • `1` – доступ есть; • `0` – доступа нет
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Пример

```
$rv = $dbh->ping();
if ($rv != 1)
{
    print "LINTER is not available.\n";
    return;
```



```
}
```

Выполнение методов DBD-драйвера (func)

Назначение

Метод `func` осуществляет выполнение методов, присущих только DBD-драйверу СУБД ЛИНТЕР.

Пакет

Package Linter::dr

Прототип

```
$drh->func(@function_paramaters,'function_name');
```

Получить код завершения последнего метода (err)

Назначение

Методом `Err` можно получить код завершения последнего выполненного метода.

Пакет

Package Linter::dr

Прототип

Этот метод доступен через вызов метода `func`.

```
$drh->func('err');
```

Возвращаемое значение

Метод возвращает код завершения последней операции.

Прямое обращение к базе данных (linter)

Назначение

Метод `Linter` осуществляет прямое обращение к БД с помощью внутреннего (Call) интерфейса СУБД ЛИНТЕР.

Пакет

Package Linter::dr

Прототип

```
$rc = $drh->
func(\%CBL,\$arg2,\$arg3,\$arg4,\$arg5,'linter');
```

Значения полей контрольного блока обмена (CBL) зависят от выполняемой команды СУБД ЛИНТЕР (см. документ «СУБД ЛИНТЕР. Интерфейс нижнего уровня»).

Поля CBL можно заполнять двумя способами:

Первый способ:

```
%CBL=(
'CodErr'=>\$coderr,
'Prior'=>\$prior,
'NumChan'=>\$numchan,
'UserName'=>\$username,
'Command'=>\$command,
'Node'=>\$node,
'RowId'=>\$rowid,
'RowCount'=>\$rowcount,
'PrzExe'=>\$przexe,
'SysErr'=>\$syserr,
'LnBufRow'=>\$lnbufrow,
'Reserve'=>\$reserve,
);
```

Второй способ:

```
$CBL{'NumChan'}=\$numchan;
$CBL{'Command'}=\$command;
$CBL{'PrzExe'}=\$przexe;
$CBL{'LnBufRow'}=\$lnbufrow;
...
```

Создание строкового литерала (qoute)

Назначение

Метод `quote` возвращает экранированный в соответствии с правилами языка SQL СУБД ЛИНТЕР строковый литерал для последующего его использования в качестве элемента генерируемого клиентским приложением текста SQL-запроса.

Пакет

Package Linter::db

Прототип

```
$sql = $dbh->quote($value);
$sql = $dbh->quote($value, $data_type);
```

Параметр	Описание
$value	Строковый Perl-литерал
$data_type	Тип данных символьного представления возвращаемого литерала

Возвращаемые значения

Переменная	Описание
`$sql`	Строковый литерал, экранированный в соответствии с языком SQL СУБД ЛИНТЕР

Переменная	Описание
$DBI::errstr	Диагностическое сообщение (в случае ошибки)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
$dbh->do("create or replace table TEST_TABLE(inches
 varchar(16));");
my $sql = sprintf("insert into TEST_TABLE values (%s);",
                  $dbh->quote("15''"));
$dbh->do($sql);
my $sth = $dbh->prepare("select * from TEST_TABLE");
$sth->execute();
my $data = $sth->fetchrow_arrayref();
print($data->[0]);
$dbh->do("drop table TEST_TABLE;");
$sth->finish();
$dbh->disconnect();
```

Создание идентификатора объекта БД (quote_identifier)

Назначение

Метод quote_identifier возвращает идентификатор объекта БД ЛИНТЕР для последующего его использования в качестве элемента генерируемого клиентским приложением текста SQL-запроса.

Пакет

Package Linter::db

Прототип

```
$sql = $dbh->quote_identifier($name);
$sql = $dbh->quote_identifier($catalog, $schema, $table, \%attr );
```

Параметр	Описание
$name	Строковый Perl-литерал
$catalog	Параметр не поддерживается
$schema	Идентификатор схемы
$table	Идентификатор таблицы
\%attr	Параметр не поддерживается

Возвращаемые значения

Переменная	Описание
$sql	Идентификатор объекта БД, экранированный в соответствии с языком SQL СУБД ЛИНТЕР
$DBI::errstr	Диагностическое сообщение (в случае ошибки)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
my $quoted_table = $dbh->quote_identifier("MAIN TABLE");
my $quoted_value = $dbh->quote("20''");
my $crt_sql = sprintf("create or replace table %s(inches
 varchar(16));", $quoted_table);
my $ins_sql = sprintf("insert into %s values (%s);",
 $quoted_table, $quoted_value);
my $sel_sql = sprintf("select * from %s;", $quoted_table);
my $del_sql = sprintf("drop table %s;", $quoted_table);

$dbh->do($crt_sql);
$dbh->do($ins_sql);
my $sth = $dbh->prepare($sel_sql);
$sth->execute();
my $data = $sth->fetchrow_arrayref();
print($data->[0]);
$dbh->do($del_sql);
$sth->finish();
$dbh->disconnect();
```

Подготовка SQL-оператора к выполнению (prepare)

Назначение

Метод `prepare` выполняет подготовку SQL-оператора к выполнению и создает объект класса «предложение».

Пакет

Package Linter::db

Прототип

```
$sth = $dbh->prepare($statement);
```

Параметр	Описание
`$statement`	Текст SQL-оператора, заканчивающийся знаком «;». SQL-оператор может включать неименованные и/или именованные динамические параметры (см. документ «СУБД ЛИНТЕР. Справочник по SQL», раздел «SQL-операторы с параметрами»)

Возвращаемые значения

Переменная	Описание
`$sth`:	
описатель предложения	Нормальное завершение
undef	Ошибка подготовки предложения

Переменная	Описание
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки подготовки предложения)

Создание кэшированного претранслированного запроса (prepare_cached)

Назначение

Метод `prepare_cached` выполняет подготовку запроса к выполнению и сохраняет дескриптор подготовленного запроса (экземпляр класса Statement), ассоциированный с соединением, в кэше запросов. Если другой вызов метода `prepare_cached` выполняется с тем же исходным текстом запроса, то используется ранее подготовленный запрос из кэша и обращение к СУБД не осуществляется.

Пакет

Package Linter::db

Прототип

```
$sth = $dbh->prepare_cached($statement);
$sth = $dbh->prepare_cached($statement, \%attr, $if_active);
```

Параметр	Описание
`$statement`	Текст SQL-запроса (возможно, с параметрами)
`\%attr`	Параметр игнорируется
`$if_active`	Управление поведением при наличии активного кэшированного экземпляра Statement: • `0` – генерирует предупреждение, вызывает метод `finish` перед возвращением экземпляра Statement (поведение по умолчанию); • `1` – вызывает метод `finish`, выдача предупреждения не осуществляется; • `2` – отключает любые проверки; • `3` – удаляет существующий экземпляр Statement из кэша и вставляет новый экземпляр на его место

Возвращаемые значения

Переменная	Описание
`$dbh`	Ссылка на дескриптор кэша претранслированного запроса или undef при возникновении ошибки
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки выполнения метода)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
```

```
my $sth = $dbh->prepare_cached("insert into TEST_TABLE values
 (?, ?);");
```

Подтверждение изменений в базе данных (commit)

Назначение

Метод `commit` осуществляет подтверждение изменений в БД, произведенных последней выполнявшейся транзакцией.

Пакет

Package Linter::db

Прототип

```
$rc = $dbh->commit();
```

Возвращаемые значения

Переменная	Описание
`$rc`:	
`1`	Нормальное завершение
`0`	Ошибка обработки транзакции
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Отмена изменений в базе данных (rollback)

Назначение

Метод `Rollback` выполняет отмену изменений в БД, произведенных последней выполнявшейся транзакцией.

Пакет

Package Linter::db

Прототип

```
$rc = $dbh->rollback([$savepoint]);
```

Примечание
В текущей версии откат транзакции к точкам сохранения не поддерживается.

Возвращаемые значения

Переменная	Описание
`$rc`:	
`1`	Нормальное завершение
`0`	Ошибка обработки транзакции
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Включение транзакционного режима (begin_work)

Назначение

Метод `begin_work` устанавливает режим управляемой фиксации изменений, переводя атрибут `AutoCommit` в значение `0`. В случае если атрибут `AutoCommit` был уже отключен, будет возвращена ошибка. После вызова `commit` или `rollback` атрибут `AutoCommit` будет автоматически включен (значение `1`), тем самым задействуется режим автоматической фиксации изменений.

Пакет

Package Linter::db

Прототип

```
$rc = $dbh->begin_work();
```

Возвращаемые значения

Переменная	Описание
`$rc`:	
`1`	Нормальное завершение
`0`	Ошибка обработки транзакции
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Выполнить SQL-предложение (do)

Назначение

Метод `do` осуществляет выполнение SQL-предложения (кроме `execute procedure`).

Пакет

Package Linter::db

Прототип

```
$rv = $dbh->do($statement, \%attributes, @values);
```

Параметр	Описание
`$statement`	SQL-предложение
`%attributes`	Зарезервировано для будущего применения.
`@values`	Массив значений динамических параметров (если SQL-предложение содержит параметры)

Возвращаемые значения

Переменная	Описание
`$rv`:	
количество обработанных записей	Нормальное завершение

Переменная	Описание
значение '0E0'	Нет обработанных записей
undef	Ошибка обработки SQL-предложения
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Вывод результата выборки данных (dump_results)

Назначение

Метод `dump_results` выводит содержимое выборки данных в удобном для просмотра и анализа виде. Метод используется, в основном, для тестирования SQL-запросов при отладке клиентских приложений. Метод `dump_results` вызывает метод `neat_list` для форматирования записей выборки данных.

Пакет

Package Linter::db

Прототип

```
$rows = $sth->dump_results($maxlen, $lsep, $fsep, $fh);
```

Параметр	Описание
`$maxlen`	Максимальная длина полей строки результата (по умолчанию 35)
`$lsep`	Разделитель строк результата (по умолчанию "\n")
`$fsep`	Разделитель полей строки результата (по умолчанию ",")
`$fh`	Спецификация файла, в который выводятся результаты (по умолчанию stdout)

Возвращаемые значения

Переменная	Описание
`$rows`	Содержимое выборки данных
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
my $sth = $dbh->prepare("select MAKE, MODEL from AUTO limit 10;");
my $rows = $sth->dump_results();
print("Total number of rows is $rows\n");
$sth->finish();
$dbh->disconnect();
```

Получить информацию о драйвере (get_info)

Назначение

Метод `get_info` предоставляет общие сведения о реализованных и поддерживаемых DBD-драйвером СУБД ЛИНТЕР возможностях.

Пакет

Package Linter::db

Прототип

```
$value = $dbh->get_info($info_type);
```

Параметр	Описание
`$info_type`	Числовой идентификатор запрашиваемой характеристики драйвера

Примеры предоставляемых методом `get_info` характеристик драйвера:

Условное имя характеристики	Десятичное значение	Описание	Пример
SQL_DATA_SOURCE_NAME	2	Имя источника данных	'dbi::Linter::DEMO'
SQL_USER_NAME	47	Имя пользователя	'SYSTEM'
SQL_MAX_COLUMN_NAME_LEN	30	Максимальная длина имени столбца	66

Примечание
Спецификация DBI-интерфейса предусматривает предоставление информации о более 200 характеристик, перечень которых приведен в оригинальной спецификации ODBC. В случае если DBD-драйвер СУБД ЛИНТЕР не поддерживает запрошенную характеристику, то будет возвращено значение undef.

Возвращаемые значения

Переменная	Описание
`$value`	Символьное (в кодировке ANSI) значение запрошенной характеристики или undef, если запрошенная характеристика не поддерживается

Примеры

```
$value = $dbh->get_info(2);  # SQL_DATA_SOURCE_NAME
$value = $dbh->get_info(47); # SQL_USER_NAME

Вместо явного числового значения параметра $info_type
 можно использовать значение, возвращаемое методом
 DBI::Const::GetInfoType. Например:
use DBI::Const::GetInfoType;
$value = $dbh->get_info($GetInfoType{SQL_MAX_COLUMN_NAME_LEN});
```

Получить метаданные объекта БД (table_info)

Назначение

Метод `table_info` предоставляет метаданные о табличных объектах БД.

Пакет

Package Linter::db

Прототип

```
$sth = $dbh->table_info($catalog, $schema, $table, $type);
```

Параметр	Описание
`$catalog`	Значение параметра игнорируется
`$schema`	Имя схемы (владельца) объекта БД (строка длиной не более 66 символов в ANSI-кодировке)
`$table`	Имя объекта БД (строка длиной не более 66 символов в ANSI-кодировке)
`$type`	Список из одного или более типов объектов. Допустимые значения: • "TABLE"; • "VIEW"; • "SYNONYM"; • "SYSTEM TABLE".

Синтаксис значений параметров:

1) параметры `$catalog`, `$schema`, `$table`, `$type` могут быть заданы шаблоном. Элементы шаблона:
   - символ процента ('%') обозначает любую последовательность символов. Например при использованиии шаблона '%FOO%' метод будет возвращать информацию о всех объектах заданного типа, в наименовании которых присутствуют символы 'FOO' (см. документацию «СУБД ЛИНТЕР. Справочник по SQL», раздел «Предикат подобия»);
   - символ подчеркивания ('_') обозначает любой одиночный символ. Например, при использованиии шаблона 'FOO_%' метод будет возвращать информацию о всех объектах заданного типа, наименование которых начинается с 'FOO' (т.е. будет аналогичен шаблону 'FOO%' или 'FOO_BAR%' и ему буде соответствовать имя таблицы 'FOO1BAR') (см. документацию «СУБД ЛИНТЕР. Справочник по SQL», раздел «Предикат подобия»);
2) значение параметра `$type` может быть заключено в одинарные или парные апострофы, например:

```
$type = 'TABLE';
$type = "TABLE";
$type = "'TABLE'";
```

3) значение параметра `$type` может быть в виде списка значений, разделенных запятой, например:

```
$type = 'TABLE', 'VIEW';
$type = "TABLE, VIEW";
$type = "'TABLE', 'VIEW'";
```


Особенности обработки параметров:

1) если значение параметра `$catalog` задано шаблоном '%', а значения параметров `$schema` и `$table` являются пустыми, то результат будет содержать одну строку с пустыми значениями столбцов.

```
$sth = $dbh->table_info('%', '', '');
```

2) если значение параметра `$schema` задано шаблоном '%', а значения параметров `$catalog` и `$table` являются пустыми, то результат будет содержать список имен схем.

3) если значение параметра `$type` задано шаблоном '%', а значения параметров `$catalog`, `$schema` и `$table` являются пустыми, то результат будет содержать список типов объектов БД.

Примечание
Если клиентское приложение выполняется от имени пользователя БД, который не имеет дискретного или мандатного доступа к некоторым объектам БД, то результат не будет содержать инфорацию об этих объектах.

Возвращаемые значения

Переменная	Описание
`$sth`	Объект Statement Handle

Запись метаданных об объекте БД представлена в виде массива из следующих элементов:

Имя элемента	Значение
`TABLE_CAT`	Пустая строка
`TABLE_SCHEM`	Имя схемы (символьное значение в кодировке ANSI длиной до 66 символов)
`TABLE_NAME`	Имя объекта БД
`TABLE_TYPE`	Тип объекта БД (одно из значений, перечисленных в описании параметра `$type`)
`REMARKS`	Для таблиц – описание (комментарий) таблицы (если данная информация присутствует в системной таблице `$$$OBJ_COMMENTS` или пустая строка, если комментарий отсутствует). Для объектов других типов возвращается пустая строка

Примечание
Для перемещения по записям массива можно использовать методы `fetchall_arrayref`, `fetchall_hashref` и т.п.

Пример

```
$sth = $dbh->table_info('', '%PER%', '', '');
while (my $data_ref = $sth->fetchrow_hashref())
{
    print($data_ref->{TABLE_CAT} . "\n");
```

```
    print($data_ref->{TABLE_SCHEM} . "\n");
    print($data_ref->{TABLE_NAME} . "\n");
    print($data_ref->{TABLE_TYPE} . "\n");
    print($data_ref->{REMARKS} . "\n");
}
```

Получить метаданные столбцов табличного объекта (column_info)

Назначение

Метод `column_info` предоставляет метаданные столбцов табличного объекта БД.

Пакет

Package Linter::db

Прототип

```
$sth = $dbh->column_info($catalog, $schema, $table, $column);
```

Параметр	Описание
`$catalog`	См. метод table_info
`$schema`	См. метод table_info
`$table`	См. метод table_info
`$column`	Имя столбца табличного объекта БД (строка длиной не более 66 символов в ANSI-кодировке)

Возвращаемые значения

Переменная	Описание
`$sth`	Объект Statement Handle

Запись метаданных о столбце табличного объекта БД представлена в виде массива из следующих элементов:

Имя элемента	Значение
`TABLE_CAT`	Пустая строка
`TABLE_SCHEM`	Имя схемы объекта (символьное значение в кодировке ANSI длиной до 66 символов)
`TABLE_NAME`	Имя таблицы
`COLUMN_NAME`	Имя столбца
`DATA_TYPE`	Числовой идентификатор типа данных ODBC SQL (см. таблицу 4)
`TYPE_NAME`	Имя типа данных в терминах СУБД ЛИНТЕР
`COLUMN_SIZE`	Максимальный размер для символьного представления числовых типов данных
`BUFFER_LENGTH`	Размер буфера для загрузки значения столбца

Имя элемента	Значение
DECIMAL_DIGITS	Количество десятичных цифр после запятой, используемых для представления значения числового типа данных
NUM_PREC_RADIX	Основание системы счисления: 10 для числовых типов данных и undef для остальных
NULLABLE	Индикатор допустимости NULL-значений. Возможные значения: • SQL_NO_NULLS – 0 (NULL-значения не разрешены); • SQL_NULLABLE – 1 (NULL-значения разрешены)
REMARKS	Описание (комментарий) столбца (если данная информация присутствует в системной таблице $$$OBJ_COMMENTS или пустая строка, если комментарий отсутствует)
COLUMN_DEF	Значение столбца по умолчанию (или undef, если значение по умолчанию не установлено)
SQL_DATA_TYPE	Числовой идентификатор типа данных ODBC SQL (см. таблицу 4)
SQL_DATETIME_SUB	Целое число, описывающее подтип данных DATE
CHAR_OCTET_LENGTH	Максимальная длина в байтах символьного или байтового типа данных столбца
ORDINAL_POSITION	Порядковый номер столбца в таблице (отсчет начинается с 1)
IS_NULLABLE	Индикатор допустимости NULL-значения. Возможные значения: 'NO', 'YES', ''

Примечание

Если клиентское приложение выполняется от имени пользователя БД, который не имеет дискретного или мандатного доступа к некоторым объектам БД, то результат не будет содержать инфорацию об этих объектах.

Таблица 4. Соответствие типов данных СУБД ЛИНТЕР и типов данных ODBC SQL

Имя типа данных СУБД ЛИНТЕР	Идентификатор типа данных ODBC SQL	Числовой идентификатор ODBC SQL
CHARACTER	SQL_CHAR	1
SMALLINT	SQL_SMALLINT	5
INTEGER	SQL_INTEGER	4
BIGINT	SQL_BIGINT	-5
REAL	SQL_REAL	7
DOUBLE	SQL_DOUBLE	8
DATE	SQL_TYPE_TIMESTAMP	93
DECIMAL	SQL_DECIMAL	3
BYTE	SQL_BINARY	-2
BLOB	SQL_LONGVARBINARY	-4

Имя типа данных СУБД ЛИНТЕР	Идентификатор типа данных ODBC SQL	Числовой идентификатор ODBC SQL
VARCHAR	SQL_VARCHAR	12
VARBYTE	SQL_VARBINARY	-3
BOOLEAN	SQL_BIT	-7
NCHAR	SQL_WCHAR	-8
NCHAR VARYING	SQL_WVARCHAR	-9
EXTFILE	SQL_CHAR	1

Примечание
Для перемещения по записям массива можно использовать метод `fetchall_arrayref`, `fetchall_hashref` и т.п.

Примеры

Получить информацию о столбцах таблицы PERSON:

```
$sth = $dbh->column_info('', '', 'PERSON', '');
while (my $data_ref = $sth->fetchrow_hashref())
{
    print($data_ref->{TABLE_CAT} . "\n");
    print($data_ref->{TABLE_SCHEM} . "\n");
    print($data_ref->{TABLE_NAME} . "\n");
    print($data_ref->{COLUMN_NAME} . "\n");
    print($data_ref->{DATA_TYPE} . "\n");
    print($data_ref->{TYPE_NAME} . "\n");
    print($data_ref->{COLUMN_SIZE} . "\n");
    print($data_ref->{BUFFER_LENGTH} . "\n");
    print($data_ref->{DECIMAL_DIGIT} . "\n");
    print($data_ref->{NUM_PREC_RADIX} . "\n");
    print($data_ref->{NULLABLE} . "\n");
    print($data_ref->{REMARKS} . "\n");
    print($data_ref->{COLUMN_DEF} . "\n");
    print($data_ref->{SQL_DATA_TYPE} . "\n");
    print($data_ref->{SQL_DATETIME_SUB} . "\n");
    print($data_ref->{CHAR_OCTET_LENGTH} . "\n");
    print($data_ref->{ORDINAL_POSITION} . "\n");
    print($data_ref->{IS_NULLABLE} . "\n");
}
```

Получить метаданные первичных ключей табличного объекта (primary_key_info)

Назначение

Метод `primary_key_info` предоставляет информацию о первичных ключах табличного объекта БД.

Пакет

Package Linter::db

Прототип

```
$sth = $dbh->primary_key_info($catalog, $schema, $table);
```

Параметр	Описание
$catalog	См. метод table_info (параметр не принимает шаблон для поиска)
$schema	См. метод table_info (параметр не принимает шаблон для поиска)
$table	См. метод table_info (параметр не принимает шаблон для поиска)

Возвращаемые значения

Переменная	Описание
$sth	Объект Statement Handle

Запись метаданных о первичном ключе табличного объекта БД представлена в виде массива из следующих элементов:

Имя элемента	Значение
TABLE_CAT	Пустая строка
TABLE_SCHEM	Имя схемы таблицы (символьное значение в кодировке ANSI длиной до 66 символов)
TABLE_NAME	Имя таблицы
COLUMN_NAME	Имя столбца, который является первичным ключом
KEY_SEQ	Значение 1
PK_NAME	Имя первичного ключа

Примечание
Если клиентское приложение выполняется от имени пользователя БД, который не имеет дискретного или мандатного доступа к некоторым объектам БД, то результат не будет содержать инфорацию об этих объектах.

Пример

Получить информацию о первичных ключах таблицы TEST_TABLE:

```
my $drh = DBI->install_driver('Linter');
my $dbh = $drh->connect("", "SYSTEM", "MANAGER8")
or die "Could not connect to database: " . DBI->errstr;
$dbh->do("create table TEST_TABLE(int_column int, char_column
 char(16), pk_col int primary key);");
my $sth = $dbh->primary_key_info('', 'TEST_TABLE', '');
while (my $data_ref = $sth->fetchrow_hashref())
{
    print($data_ref->{TABLE_CAT} . "\n");
```

```
    print($data_ref->{TABLE_SCHEM} . "\n");
    print($data_ref->{TABLE_NAME} . "\n");
    print($data_ref->{COLUMN_NAME} . "\n");
    print($data_ref->{KEY_SEQ} . "\n");
    print($data_ref->{PK_NAME} . "\n");
}
$dbh->do("drop table TEST_TABLE;");
$sth->finish();
$dbh->disconnect();
```

Получить список имен столбцов, входящих в состав первичного составного ключа табличного объекта (primary_key)

Назначение

Метод `primary_key` возвращает список имен столбцов, которые входят в состав первичного составного ключа табличного объекта. Является упрощенным вариантом метода `primary_key_info`.

Пакет

Package Linter::db

Прототип

```
@key_column_names = $dbh->primary_key($catalog, $schema, $table);
```

Параметр	Описание
`$catalog`	См. метод table_info (параметр не принимает шаблон для поиска)
`$schema`	См. метод table_info (параметр не принимает шаблон для поиска)
`$table`	См. метод table_info (параметр не принимает шаблон для поиска)

Возвращаемые значения

Переменная	Описание
`@key_column_names`	Список имен столбцов

Примечание
Если клиентское приложение выполняется от имени пользователя БД, который не имеет дискретного или мандатного доступа к некоторым объектам БД, то результат не будет содержать инфорацию об этих объектах.

Пример

Получить список имен столбцов, которые являются первичными ключами в рамках составного первичного ключа таблицы TEST_TABLE:

```
my $drh = DBI->install_driver('Linter');
```

```
my $dbh = $drh->connect("", "SYSTEM", "MANAGER8")
or die "Could not connect to database: " . DBI->errstr;
$dbh->do("create table TEST_TABLE(int_column int, smallint_column
 smallint, char_column char(16));");
$dbh->do("alter table TEST_TABLE add primary key (int_column,
 smallint_column);");
my @pkeys = $dbh->primary_key('', 'TEST_TABLE', '');
for (my $i = 0; $i < scalar(@pkeys); $i++)
{
    print($pkey[$i] . "\n");
}
$dbh->do("drop table TEST_TABLE;");
$dbh->disconnect();
```

Получить метаданные внешних ключей табличного объекта (foreign_key_info)

Назначение

Метод `foreign_key_info` предоставляет информацию о внешних ключах и/или ссылках табличного объекта БД.

Пакет

Package Linter::db

Прототип

```
$sth = $dbh->foreign_key_info($pk_catalog, $pk_schema, $pk_table,
                              $fk_catalog, $fk_schema, $fk_table);
```

Параметр	Описание
`$pk_catalog`	Имя каталога табличных объектов с первичным ключом (см. метод table_info, параметр не принимает шаблон для поиска)
`$pk_schema`	Имя схемы табличных объектов с первичным ключом (см. метод table_info, параметр не принимает шаблон для поиска)
`$pk_table`	Имя табличного объекта с первичным ключом (см. метод table_info, параметр не принимает шаблон для поиска)
`$fk_catalog`	Имя каталога табличных объектов с внешним ключом (см. метод table_info, параметр не принимает шаблон для поиска)
`$fk_schema`	Имя схемы табличных объектов с внешним ключом (см. метод table_info, параметр не принимает шаблон для поиска)
`$fk_table`	Имя табличного объекта с внешним ключом (см. метод table_info, параметр не принимает шаблон для поиска)

Возвращаемые значения

Переменная	Описание
`$sth`	Объект Statement Handle

Примечание
Если клиентское приложение выполняется от имени пользователя БД, который не имеет дискретного или мандатного доступа к некоторым объектам БД, то результат не будет содержать инфорацию об этих объектах.

Особенности реализации метода:

1) если параметры обоих табличных объектов переданы и если в таблице с внешними ключами имеются ссылки на таблицы с первичными ключами, то метод возвращает внешние ключи;
2) если указаны параметры только таблицы с первичным ключом, то метод возвращает набор первичных ключей указанной таблицы и все внешние ключи, которые ссылаются на нее;
3) если указаны параметры только таблицы с внешним ключом, то метод возвращает набор внешних ключей указанной таблицы и все первичные ключи, на которые ссылаются внешние ключи.

Запись метаданных о внешних ключах табличного объекта БД представлена в виде массива из следующих элементов:

Имя элемента	Значение
PKTABLE_CAT	Пустая строка
PKTABLE_SCHEM	Имя схемы, содержащей таблицу с первичным (уникальным) ключом
PKTABLE_NAME	Имя таблицы с первичным (уникальным) ключом
PKCOLUMN_NAME	Имя столбца, являющегося первичным (уникальным) ключом
FKTABLE_CAT	Пустая строка
FKTABLE_SCHEM	Имя схемы, содержащей таблицу с внешним ключом
FKTABLE_NAME	Имя таблицы с внешним ключом.
FKCOLUMN_NAME	Имя столбца, являющегося внешним ключом
KEY_SEQ	Значение 1
UPDATE_RULE	Действие, выполняемое со значением внешнего ключа, при обновлении первичного ключа, на который ссылается внешний ключ. Возможные действия: • CASCADE: 0 (значение внешнего ключа устанавливается равным новому значению первичного ключа); • SET NULL: 2 (значению внешнего ключа присваивается NULL-значение); • NO ACTION: 3 (обновление первичного ключа запрещается с выдачей соответствующего кода завершения); • SET DEFAULT: 4 (внешнему ключу присваивается значение по умолчанию)
DELETE_RULE	Действие, выполняемое со значением внешнего ключа, при удалении первичного ключа, на который ссылается внешний ключ. Возможные действия аналогичны действиям, описанным в UPDATE_RULE
FK_NAME	Имя внешнего ключа

Имя элемента	Значение
PK_NAME	Имя первичного (уникального) ключа
DEFERRABILITY	Undef

Пример

```
my $drh = DBI->install_driver('Linter');
my $dbh = $drh->connect("", "SYSTEM", "MANAGER8")
or die "Could not connect to database: " . DBI->errstr;
$dbh->do("create table TEST_TABLE(int_column int, char_column
 char(16), pk_col int primary key);");
$dbh->do("alter table TEST_TABLE add foreign key (pk_col)
 references TEST_TABLE(pk_col);");
my $sth = $dbh->foreign_key_info('', 'TEST_TABLE', '', '', '',
 '');
while (my $data_ref = $sth->fetchrow_hashref())
{
  print($data_ref->{PKTABLE_CAT} . "\n");
  print($data_ref->{PKTABLE_SCHEM} . "\n");
  print($data_ref->{PKTABLE_NAME} . "\n");
  print($data_ref->{PKCOLUMN_NAME} . "\n");
  print($data_ref->{FKTABLE_CAT} . "\n");
  print($data_ref->{FKTABLE_SCHEM} . "\n");
  print($data_ref->{FKTABLE_NAME} . "\n");
  print($data_ref->{FKCOLUMN_NAME} . "\n");
  print($data_ref->{KEY_SEQ} . "\n");
  print($data_ref->{UPDATE_RULE} . "\n");
  print($data_ref->{DELETE_RULE} . "\n");
  print($data_ref->{FK_RULE} . "\n");
  print($data_ref->{PK_RULE} . "\n");
  print($data_ref->{DEFERRABILITY} . "\n");
}
$dbh->do("drop table TEST_TABLE;");
$sth->finish();
$dbh->disconnect();
```

Получить статистическую информацию о таблице и ее индексах (statistics_info)

Назначение

Метод statistics_info предоставляет количественные данные и позволяет получить информацию, которая может быть использована для получения статистики о таблице и ее индексах.

Пакет

Package Linter::db

Прототип

```
$sth = $dbh->statistics_info($catalog, $schema, $table,
                             $unique_only, $quick);
```

Параметр	Описание
$catalog	См. метод table_info (параметр не принимает шаблон для поиска)
$schema	См. метод table_info (параметр не принимает шаблон для поиска)
$table	См. метод table_info (параметр не принимает шаблон для поиска)
$unique_only	Задает вид предоставляемой информации об индексах: • true: только об уникальных индексах; • false: о всех индексах
$quick	Параметр игнорируется

Возвращаемые значения

Переменная	Описание
$sth	Объект Statement Handle

Метаданные индекса – массив значений следующих атрибутов индекса:

Имя элемента	Значение
TABLE_CAT	Пустая строка
TABLE_SCHEM	Имя схемы таблицы (символьное значение в кодировке ANSI длиной до 66 символов)
TABLE_NAME	Имя таблицы
NON_UNIQUE	Признак уникальности значений столбца: • 0 – для уникальных индексов; • 1 – для не уникальных индексов
INDEX_QUALIFIER	Пустая строка
INDEX_NAME	Имя индекса
TYPE	Значение 3 (соответствует идентификатору SQL_INDEX_OTHER по спецификации ODBC)
ORDINAL_POSITION	Порядковый номер данного индексированного столбца (отсчет начинается с 1)
COLUMN_NAME	Имя столбца
ASC_OR_DESC	Значение «A» – сортировка по возрастанию, пустая строка – сортировка индексов не поддерживается
CARDINALITY	Значение undef
PAGES	Значение undef
FILTER_CONDITION	Значение undef

Пример

```
my $drh = DBI->install_driver('Linter');
```



```
my $dbh = $drh->connect("", "SYSTEM", "MANAGER8")
or die "Could not connect to database: " . DBI->errstr;
$dbh->do("create table TEST_TABLE(int_column int, char_column
 char(16));");
$dbh->do("create index ndx_test_table on
 TEST_TABLE(int_column);");
my $sth = $dbh->statistics_info('', 'TEST_TABLE', '', undef,
 undef);
while (my $data_ref = $sth->fetchrow_hashref())
{
          print($data_ref->{TABLE_CAT} . "\n");
    print($data_ref->{TABLE_SCHEM} . "\n");
    print($data_ref->{TABLE_NAME} . "\n");
    print($data_ref->{NON_UNIQUE} . "\n");
    print($data_ref->{INDEX_QUALIFIER} . "\n");
    print($data_ref->{INDEX_NAME} . "\n");
    print($data_ref->{TYPE} . "\n");
    print($data_ref->{ORDINAL_POSITION} . "\n");
    print($data_ref->{COLUMN_NAME} . "\n");
    print($data_ref->{ASC_OR_DESC} . "\n");
    print($data_ref->{CARDINALITY} . "\n");
    print($data_ref->{PAGES} . "\n");
          print($data_ref->{FILTER_CONDITION} . "\n");
}
$dbh->do("drop table TEST_TABLE;");
$sth->finish();
$dbh->disconnect();
```

Получить список имен таблиц (tables)

Назначение

Метод `tables` предоставляет список имен таблиц БД. Является упрощенным вариантом метода `table_info`.

Пакет

Package Linter::db

Прототип

```
@names = $dbh->tables($catalog, $schema, $table, $type);
```

Параметр	Описание
`$catalog`	См. метод table_info
`$schema`	См. метод table_info
`$table`	См. метод table_info
`$type`	См. метод table_info

Возвращаемые значения

Переменная	Описание
`@names`	Список имен таблиц

Пример

```
$tbl = $dbh->tables('', '%PER%', '', '');
for (my $i = 0; $i < scalar(@tbl); $i++)
{
    print($tbl[$i] . "\n");
}
```

Получить описание всех поддерживаемых типов данных (type_info_all)

Назначение

Метод `type_info_all` предоставляет ссылку на массив, который содержит информацию о каждом типе данных, поддерживаемом СУБД и DBD-драйвером. Массив и его содержимое должны использоваться только для чтения.

Структура возвращаемого массива:

Порядковый номер элемента массива	Значение элемента массива
1	Ссылка на хеш-массив, содержащий имена атрибутов типов данных. Обращение к элементам хеш-массива выполняется по хеш-ключу Name => Index

```
$type_info_all = [
    {   TYPE_NAME          => 0,
        DATA_TYPE          => 1,
        COLUMN_SIZE        => 2,
        LITERAL_PREFIX     => 3,
        LITERAL_SUFFIX     => 4,
        CREATE_PARAMS      => 5,
        NULLABLE           => 6,
        CASE_SENSITIVE     => 7,
        SEARCHABLE         => 8,
        UNSIGNED_ATTRIBUTE => 9,
        FIXED_PREC_SCALE   => 10,
        AUTO_UNIQUE_VALUE  => 11,
        LOCAL_TYPE_NAME    => 12,
        MINIMUM_SCALE      => 13,
        MAXIMUM_SCALE      => 14,
        SQL_DATA_TYPE      => 15,
        SQL_DATETIME_SUB   => 16,
        NUM_PREC_RADIX     => 17,
```

Порядковый номер элемента массива	Значение элемента массива
	INTERVAL_PRECISION => 18, }
2...n	Ссылка на массив, содержащий значения перечисленных в первом элементе атрибутов типов данных. Атрибуты типов данных, предоставляемые DBD-драйвером СУБД ЛИНТЕР, приведены в таблице 5

Таблица 5. Список и порядок предоставляемых DBD-драйвером СУБД ЛИНТЕР атрибутов типов данных

Имя хеш-индекса	Номер хеш-индекса	Описание
TYPE_NAME	0	Имя типа данных, указываемое в операторе CREATE TABLE
DATA_TYPE	1	Идентификатор (номер) типа данных в терминах языка SQL
COLUMN_SIZE	2	Для числовых типов данных означает общее количество цифр (если значение NUM_PREC_RADIX равно 10). Для строковых типов данных – максимальный размер строки в символах. Для дат – максимальное количество символов, необходимых для отображения значения
LITERAL_PREFIX	3	Символ, используемый для обозначения префикса символьных литералов
LITERAL_SUFFIX	4	Символ, используемый для обозначения суффикса символьных литералов
CREATE_PARAMS	5	Имена параметров для определения типа данных. Например, значение параметра для типа DECIMAL будет "precision,scale", где точность и масштаб являются целочисленными значениями, а для типа VARCHAR (и других строковых типов) будет "length", где длина также задается целочисленным значением. Значение NULL (undef) возвращается для тех типов данных, для которых это не применимо
NULLABLE	6	Допустимость NULL-значения: • 0 или пустая строка – недопустимо; • 1 – допустимо; • 2 – неизвестно
CASE_SENSITIVE	7	Регистрозависимость в операциях сортировки и сравнения: • 0 – отсутствует; • 1 – присутствует

Имя хеш-индекса	Номер хеш-индекса	Описание
SEARCHABLE	8	Возможность использования типа данных в предложении WHERE: • 0 – не может быть использован в предложении WHERE; • 1 – может быть использован только с предикатом LIKE; • 2 – может быть использован со всеми операторами сравнения, за исключением предиката LIKE; • 3 – может быть использован в предложении WHERE с любым оператором сравнения
UNSIGNED_ATTRIBUTE	9	Признак беззнакового типа данных: • 0 – не относится к беззнаковым; • 1 – относится к беззнаковым; • NULL (undef) – не применимо к типу данных
FIXED_PREC_SCALE	10	Указывает, имеет ли тип данных фиксированную точность и масштаб. В СУБД ЛИНТЕР ни один из типов не относится к тем типам, которые имеют фиксированную точность и масштаб (значение 0)
AUTO_UNIQUE_VALUE	11	Признак того, относится ли тип данных к типам, у которых может автоматически формироваться уникальное значение: • 0 – не относится; • 1 – относится; • NULL (undef) – не применимо для типа данных
LOCAL_TYPE_NAME	12	Имя типа данных в терминах СУБД ЛИНТЕР
MINIMUM_SCALE	13	Минимальный масштаб значений типа данных. Значение NULL (undef) возвращается для тех типов данных, для которых это не применимо
MAXIMUM_SCALE	14	Максимальный масштаб значения типа данных. Значение NULL (undef) возвращается для тех типов данных, для которых это не применимо
SQL_DATA_TYPE	15	Идентификатор (номер) типа данных в терминах языка SQL. Значение данного столбца является таким же, как в столбце DATA_TYPE, за исключением типов данных, относящихся к типу DATE

Имя хеш-индекса	Номер хеш-индекса	Описание
SQL_DATETIME_SUB	16	Целое число, описывающее подтип данных типа DATE
NUM_PREC_RADIX	17	Основание системы счисления: 10 для числовых типов данных и undef для остальных. Для точных числовых типов NUM_PREC_RADIX содержит значение 10, а значение COLUMN_SIZE содержит общее количество цифр. Значение NULL (undef) возвращается для тех типов данных, для которых это не применимо
INTERVAL_PRECISION	18	Диапазон точности для интервальных типов данных. Значение NULL (undef) возвращается для тех типов данных, для которых это не применимо

Для некоторых типов данных в возвращаемом массиве может быть представлено несколько строк массива, если имя типа данных имеет синонимы или разные варианты реализации (например, INTEGER и INTEGER AUTOINC).

Примечание
Метод `type_info_all` обычно не используется напрямую. Более удобный и полезный интерфейс для получения информации о типах данных предоставляет метод `type_info`.

Пакет

Package Linter::db

Прототип

```
$type_info_all = $dbh->type_info_all;
```

Возвращаемые значения

Переменная	Описание
`$type_info_all`	Ссылка на массив с атрибутами всех поддерживаемых СУБД ЛИНТЕР типов данных
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
or die "Could not connect to database: " . DBI->errstr;
my $tia = $dbh->type_info_all();
for (my $i = 1; $i <= scalar(@$tia); $i++)
{
    print(join("\n", @{$tia->[$i]}));
}
$dbh->disconnect();
```

Получить описание указанных типов данных (type_info)

Назначение

Метод type_info предоставляет список ссылок на хеш-массивы, содержащие информацию об указанных типах данных СУБД ЛИНТЕР.

Если параметр $data_type не определен или принимает значение SQL_ALL_TYPES, то метод предоставит полный список возможных типов данных, поддерживаемых СУБД ЛИНТЕР. В этом случае выполнение метода type_info идентично методу type_info_all.

Если параметр $data_type является ссылкой на массив, то type_info возвращает информацию для первого указанного типа данных в этом массиве.

Ключи хеш-массива являются регистрозависимыми.

Содержимое хеш-массива приведено в таблице 5.

Пакет

Package Linter::db

Прототип

```
@type_info = $dbh->type_info($data_type);
```

Параметр	Описание
$data_type	Идентификатор одного типа данных или ссылка на массив идентификаторов типов данных

Возвращаемые значения

Переменная	Описание
@type_info	Список ссылок на хеш-массивы со значениями атрибутов указанного типа данных
$DBI::errstr	Диагностическое сообщение (в случае ошибки)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
or die "Could not connect to database: " . DBI->errstr;
my $ti = $dbh->type_info(DBI::SQL_LONGVARBINARY);
print(join("\n", @{$ti->[$i]}));
$dbh->disconnect();
```

Получить идентификатор последней добавленной записи (last_insert_id)

Назначение

Метод last_insert_id возвращает идентификатор последней добавленной записи в табличный объект БД.

Пакет

Package Linter::db

Прототип

```
$rv = $dbh->last_insert_id($catalog, $schema, $table, $field
 [, \%attr]);
```

Параметр	Описание
$catalog	Параметр игнорируется
$schema	Имя схемы таблицы
$table	Имя таблицы
$field	Параметр игнорируется
%attr	Параметр игнорируется

Особенности выполнения метода:

1) значение может быть доступно после выполнения команд DML (INSERT, UPDATE, DELETE);
2) если не задано имя таблицы, то будет возвращено значение последней добавленной записи по каналу, в котором выполняется данный метод.

Примечание
Для дополнительной информации см. документацию «СУБД ЛИНТЕР. Справочник по SQL», раздел «Спецификация значений», параметр LAST_ROWID.

Возвращаемые значения

Переменная	Описание
$rv	Значение последней добавленной записи
undef	Нет данных

Пример

```
my $drh = DBI->install_driver('Linter');
my $dbh = $drh->connect("", "SYSTEM", "MANAGER8")
or die "Could not connect to database: " . DBI->errstr;
$dbh->do("create table TEST_TABLE(int_column int);");
for (my $i = 0; $i < 10; $i++)
{
    $dbh->do("insert into TEST_TABLE values ($i);");
}
my $id = $dbh->last_insert_id('', '', 'TEST_TABLE', '');
print("last insert id = $id\n");
$dbh->do("drop table TEST_TABLE;");
$dbh->disconnect();
```

Сохранить данные (snap)

Назначение

Метод `snap` выполняет принудительное сохранение данных на диск для повышения надежности.

Примечание
СУБД ЛИНТЕР обрабатываемые данные хранит в своем внутреннем буфере и выгружает их на диск по мере его заполнения. Метод `snap` заставляет СУБД выгрузить данные на диск не дожидаясь заполнения буфера.

Пакет

Package Linter::db

Прототип

```
$rc = $dbh->func($database, 'snap');
```

Параметр	Описание
`$database`	Имя базы данных

Возвращаемые значения

Переменная	Описание
`$rc`:	
`1`	Нормальное завершение
`0`	Ошибка выполнения метода

Привязка формального параметра хранимой процедуры (bind_param_inout)

Назначение

Метод `bind_param_inout` выполняет привязку значения указанного формального параметра хранимой процедуры.

Метод позволяет задать значение для inout-параметра хранимой процедуры, в которую необходимо разместить вычисленное хранимой процедурой значение выходного параметра. Значение привязываемого параметра не копируется, а только считывается в момент выполнения запроса.

Пакет

Package Linter::st

Прототип

```
$rc = $sh->bind_param_inout($p_num, \$bind_value, $max_len);
```

Параметр	Описание
`$p_num`	Порядковый номер привязываемого параметра. Отсчет начинается с `1`

Параметр	Описание
$bind_value	Ссылка на переменную, содержащую значение привязываемого параметра
$max_len	Минимальный объем памяти, выделяемый для размещения значения выходного привязываемого параметра

Возвращаемые значения

Переменная	Описание
$rc:	
1	Нормальное завершение
0	Ошибка выполнения метода
$DBI::errstr	Диагностическое сообщение (в случае ошибки)

Пример

```
my $a_val = 5;
my $b_val = 10;
my $c_val = 20;
my $dbh = $drh -> connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
$dbh->do("create procedure TEST_PROC (in a int; inout b int; inout
 c int)
          result int code b := a; c := a + b; return c; end;");
my $sth = $dbh->prepare("execute TEST_PROC(?, ?, ?);");
$sth->bind_param(1, $a_val);
$sth->bind_param_inout(2, \$b_val, 4); #4 bytes for int type
$sth->bind_param_inout(3, \$c_val, 4); #4 bytes for int type
$sth->execute();
print("a = $a_val\n");
print("b = $b_val\n");
print("c = $c_val\n");
$dbh->do("drop procedure TEST_PROC;");
$sth->finish();
$dbh->disconnect();
```

Выполнение хранимой процедуры (proc)

Назначение

Метод proc осуществляет выполнение хранимой процедуры.

Пакет

Package Linter::db

Прототип

```
$rv = $dbh->func($proc_name, @proc_parameters, 'proc');
```

```
@proc_parameters ::=
([$input_parameter,]
[\$input_output_paramater,]
[\$output_parameter])
```

Параметр	Описание
`$proc_name`	Имя хранимой процедуры
`@proc_parameters`	Массив параметров процедуры: • `$input_parameter` (параметры типа IN) • `$input_output_paramater` (параметры типа INOUT) • `$output_parameter` (параметры типа OUT)

Возвращаемые значения

Переменная	Описание
`$rv`:	
скалярное значение или объект типа «предложение»	Нормальное завершение
undef	Ошибка выполнения процедуры
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Привязка формальных параметров (bind_param)

Назначение

Метод `bind_param` выполняет привязку формальных параметров SQL-оператора к значениям.

Пакет

Package Linter::st

Прототип

```
$rc = $sth->bind_param($p_num, $bind_value [, $bind_type]);
```

Параметр	Описание
`$p_num`	Порядковый номер привязываемого параметра. Нумерация начинается с `1`
`$bind_value`	Значение привязываемого параметра (в том числе допускается и NULL-значение)
`$bind_type`	Тип привязываемого параметра. Зарезервировано для дальнейшего использования

Возвращаемые значения

Переменная	Описание
`$rc`:	

Переменная	Описание
1	Нормальное завершение
0	Ошибка выполнения метода
$DBI::errstr	Диагностическое сообщение (в случае ошибки)

Привязка формального параметра к столбцу (bind_col)

Назначение

Метод bind_col выполняет привязку параметра к возвращаемому SELECT-запросом значению столбца (полю записи выборки данных). При перемещении по выборке данных значение привязанного параметра автоматически изменяется.

Метод bind_col необходимо использовать после вызова метода execute.

Пакет

Package Linter::st

Прототип

```
$rc = $sh->bind_col($column_number, \$var_to_bind);
$rc = $sh->bind_col($column_number, \$var_to_bind, \%attr);
$rc = $sh->bind_col($column_number, \$var_to_bind, $bind_type);
```

Параметр	Описание
$column_number	Порядковый номер столбца выборки данных. Отсчет начинается с 1
$var_to_bind	Ссылка на переменную для привязки
$bind_type	Тип привязываемого параметра. Зарезервировано для дальнейшего использования

Возвращаемые значения

Переменная	Описание
$sh:	
1	Нормальное завершение
0	Ошибка выполнения метода
$DBI::errstr	Диагностическое сообщение (в случае ошибки)

Пример

```
my ($model, $make, $year);
my $dbh = $drh -> connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
my $sth = $dbh->prepare("select MODEL, MAKE, YEAR from AUTO;");
$sth->execute();
$sth->bind_col(1, \$model);
$sth->bind_col(2, \$make);
```

```
$sth->bind_col(3, \$year);
while ($sth->fetch)
{
    print("MODEL = $model\n");
    print("MAKE = $make\n");
    print("YEAR = $year\n");
}
$sth->finish();
$dbh->disconnect();
```

Привязка формальных параметров ко всем столбцам выборки данных (bind_columns)

Назначение

Метод `bind_columns` выполняет привязку параметров для каждого столбца возвращаемой SELECT-запросом выборки данных. При перемещении по выборке значения привязанных параметров автоматически изменяются.

Метод `bind_columns` необходимо использовать после вызова метода `execute()`.

Количество связываемых переменных должно соответствовать количеству столбцов в выборке данных. При несовпадении количества привязываемых параметров и количества столбцов выборки данных будет возвращена ошибка.

Пакет

Package Linter::st

Прототип

```
$rc = $sth->bind_columns(@list_of_refs_to_vars_to_bind);
```

Параметр	Описание
`@list_of_refs_to_vars_to_bind`	Список имен переменных, в которые будут загружаться значения столбцов выборки данных

Возвращаемые значения

Переменная	Описание
`$rc`:	
`1`	Нормальное завершение
`0`	Ошибка выполнения метода
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Пример

```
my ($model, $make, $year);
my $dbh = $drh -> connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
```

```
my $sth = $dbh->prepare("select MODEL, MAKE, YEAR from AUTO;");
$sth->execute();
$sth->bind_columns(\($model, $make, $year));
while ($sth->fetch)
{
    print("MODEL = $model\n");
    print("MAKE = $make\n");
    print("YEAR = $year\n");
}
$sth->finish();
$dbh->disconnect();
```

Привязка массива формальных параметров (bind_param_array)

Назначение

Метод `bind_param_array` выполняет привязку массива значений к претранслированному SQL-запросу для последующего его выполнения с помощью метода `execute_array`.

Ограничения метода:

1) нельзя использовать для передачи списка привязываемых значений в выражение типа SELECT foo WHERE bar IN (?);
2) привязка массива параметров поддерживается только для SQL-операторов INSERT, UPDAT или DELETE;
3) не допускается смешение методов `bind_param_array` и `bind_param` при привязке параметров одного и того же претранслированного SQL-запроса;
4) метод `bind_param_array` должен использоваться совместно только с методом `execute_array`.

Пакет

Package Linter::st

Прототип

```
$rc = $sh->bind_param_array($p_num, $array_ref_or_value);
$rc = $sh->bind_param_array($p_num, $array_ref_or_value,
 $bind_type);
```

Параметр	Описание
`$p_num`	Порядковый номер привязываемого параметра. Отсчет начинается с `1`
`$array_ref_or_value`	Значение или ссылка на массив значений привязываемых параметров (допускается NULL-значение параметра)
`$bind_type`	Тип привязываемого параметра. Зарезервировано для дальнейшего использования

Возвращаемые значения

Переменная	Описание
`$rc`:	
`1`	Нормальное завершение
`0`	Ошибка выполнения метода
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Пример

```
my $dbh = $drh -> connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
$dbh->do("create or repalce table TEST_TABLE
          (int_col int, char_col char(16), bigint_col bigint);");
my $sth = $dbh->prepare("insert into TEST_TABLE
 values(?, ?, ?);");
$sth->bind_param_array(1, [ 111, 222, 333 ]);
$sth->bind_param_array(2, [ 'Fisrt', 'Second', 'Third']);
$sth->bind_param_array(3, [ 111111, 222222, 333333]);
$sth->execute_array({ ArrayTupleStatus => \my @tuple_status });
$sth->finish();
$dbh->disconnect();
```

Выполнение подготовленного SQL-оператора (execute)

Назначение

Метод `execute` выполняет подготовленный методом `prepare` SQL-оператор.

Пакет

Package Linter::st

Прототип

```
$rc = $sth->execute([@paramaters_array]);
```

Параметр	Описание
`@paramaters_array`	Массив значений параметров, привязываемых к формальным параметрам SQL-предложения

Возвращаемые значения

Переменная	Описание
`$rc`:	
количество обработанных записей	Нормальное завершение
значение '0E0'	Нет обработанных записей

Переменная	Описание
undef	Ошибка обработки SQL-предложения
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Выполнение претранслированого запроса с наборами параметров (execute_array)

Назначение

Метод `execute_array` выполняет претранслированный запрос для каждого набора параметров (группы значений), привязанных либо через массив `@bind_values`, либо через вызов метода `bind_param_array`, либо через параметр `%attr`.

Подобно методу `execute` метод `execute_array` при успешном выполнении всегда возвращает true независимо от количества наборов привязываемых параметров, даже если их нет. Если при выполнении метода были выявлены ошибки, то в массиве `ArrayTupleStatus` можно найти дополнительную информацию о том, с каким набором привязанных параметров зафиксирована ошибка.

Метод может возвращать два скалярных значения в зависимости от варианта исполнения метода (скалярный или списочный):

1) `$tuples` – количество наборов значений, использованных для привязки параметров;
2) `$rows` – общее количество реально обработанных записей при выполнении запроса с данным параметром (или `-1`, если драйвер не смог определить это значение).

Значения привязываемых параметров могут быть переданы методу тремя способами:

1) построчно с помощью атрибута `ArrayTupleFetch`;
2) по столбцам путем задания привязываемых значений в параметре `@bind_values`;
3) по столбцам с помощью предшествующего вызова метода `bind_param_array`.

Если количество элементов в массиве привязываемых значений меньше количества привязываемых столбцов, то столбцы, для которых нет соответствующих значений в массиве привязываемых значений, получают undef-значения.

Атрибут `ArrayTupleFetch` позволяет задать функцию (подпрограмму), которая будет поставщиком привязываемых значений для каждого выполнения запроса. Для удобства атрибут может быть использован для указания обработчика, выполняющего вызов метода `fetchrow_arrayref` для привязки значений к каждому обрабатываемому запросу.

Атрибут `ArrayTupleStatus` может быть использован для указания ссылки на массив, в котором будут содержаться результаты выполнения запросов с каждым набором привязанных параметров. Каждый элемент массива, соответствующий порядковому номеру набора привязанных данных, может содержать следующие данные:

- количество реально обработанных записей с привязанными значениями;
- -1, если драйвер не смог определить количество реально обработанных записей;
- ссылку на массив, содержащий значения `err`, `errstr` и `state`, установленные при неуспешном выполнении.

Пакет

Package Linter::st

Прототип

```
$tuples = $sth->execute_array(\%attr);

$tuples = $sth->execute_array(\%attr, @bind_values);

($tuples, $rows) = $sth->execute_array(\%attr);

($tuples, $rows) = $sth->execute_array(\%attr, @bind_values);
```

Параметр	Описание
`\%attr`	Массив с атрибутами для указания дополнительных инструкций по выполнению метода. Возможные атрибуты: `ArrayTupleFetch` и `ArrayTupleStatus` (см. выше описание метода)
`@bind_values`	Массив элементов, каждый из которых является набором значений привязываемых к запросу параметров

Возвращаемые значения

Переменная	Описание
`$tuples`	Количество наборов значений, использованных для привязки параметров, при отсутствии наборов значений – "0E0"
`$rows`	Общее количество реально обработанных записей при выполнении запроса с данным параметром (или `-1`, если драйвер не смог определить это значение)
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
or die "Could not connect to database: " . DBI->errstr;
my @int_vals = (123, 345, 678);
my @str_vals = ("first", "second", "third");
$dbh->do("create or replace TEST_TABLE (int_col int,
                                        varchar_col
 varchar(16));");
my $sth = $dbh->prepare("insert into TEST_TABLE values (?, ?);");
my $tuples = $sth->execute_array({ ArrayTupleStatus => \my
 @tuple_status },
                                  \@int_vals, \@str_vals, );
if (!$tuples)
{
    for my $tuple (0..@tuple_status - 1)
    {
        my $status = $tuple_status[$tuple];
```



```
        $status = [0, "Skipped"] unless defined $status;
        next unless ref $status;
        printf("Failed to insert (%s, %s): %s\n",
 $int_vals[$tuple],
               $str_vals[$tuple], $status->[1]);
    }
}
$sth->finish();
$dbh->disconnect();
```

Выполнение параметрического запроса с подгружаемыми наборами привязываемых значений (execute_for_fetch)

Назначение

Метод `execute_for_fetch` используется для выполнения каких-либо массовых операций и часто используется в совокупности с методом `execute_array`, однако может применяться и самостоятельно. Основное различие между методами `execute_array` и `execute_for_fetch` в способе привязки параметров: первый метод допускает постолбцовое или построчное привязывание параметров, а второй – только построчное. Для получения значений привязываемых параметров метод вызывает специальную подпрограмму, которая и предоставляет требуемый набор значений.

Метод `execute_for_fetch` вызывает циклически подпрограмму, задаваемую параметром `$fetch_tuple_sub`, без передачи ей каких-либо аргументов до тех пор, пока она не вернет значение false (это означает, что подпрограмма исчерпала предоставление наборов привязываемых значений). Каждый возвращенный подпрограммой набор подставляемых значений используется для привязки значений с помощью вызова $sth->execute(@$tuple).

В скалярном варианте исполнения метод `execute_for_fetch` возвращает либо undef при наличии ошибок, либо количество наборов значений, использованных им для привязки параметров. Подобно методам `execute` и `execute_array`, метод `execute_for_fetch` возвращает ноль как "0E0" (false возвращается в случае ошибки). Для локализации и детализации ошибок, возникших в процессе исполнения метода, служит массив `@tuple_status`, в котором содержится информация о том, какие из наборов значений привели к ошибкам. Данный массив содержит один элемент для каждого набора подставляемых значений. Если метод `execute` выполнен успешно, то значение этого элемента совпадает с возвращенным методом `execute` значением (количество реально обработанных записей). Если выполнение метода `execute` завершилось неуспешно, то значением этого элемента является ссылка на массив с детальной информацией об ошибке ($sth->err, $sth->errstr, $sth->state).

В списочном варианте исполнения метод `execute_for_fetch` возвращает два скалярных значения: количество наборов значений, использованных им для привязки параметров, и суммарное количество записей, реально обработанных SQL-запросом для каждого набора подставленных значений (или -1, если драйвер не смог определить это значение).

Для более эффективного выполнения метода рекомендуется использовать ссылку на набор подставляемых значений ($sth->execute(@$tuple_array_ref)).

Пакет

Package Linter::st

Прототип

```
$tuples = $sth->execute_for_fetch($fetch_tuple_sub);
$tuples = $sth->execute_for_fetch($fetch_tuple_sub,
 \@tuple_status);

($tuples, $rows) = $sth->execute_for_fetch($fetch_tuple_sub);
($tuples, $rows) = $sth->execute_for_fetch($fetch_tuple_sub,
 \@tuple_status);
```

Параметр	Описание
`$fetch_tuple_sub`	Подпрограмма, предоставляющая наборы подставляемых значений
`\@tuple_status`	Ссылка на массив для размещения информации об ошибках

Возвращаемые значения

Переменная	Описание
`$tuples`	Количество наборов значений, использованных для привязки параметров
`$rows`	Суммарное количество записей, реально обработанных для каждого набора подставленных значений
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
or die "Could not connect to database: " . DBI->errstr;
my $sth_slct = $dbh->prepare("select MODEL, MAKE, YEAR from
 AUTO;");
$sth_slct->execute();
my $sth_ins = $dbh->prepare("insert into AUTO_RESERVE(MODEL, MAKE,
 YEAR)
                                values (?, ?, ?)");
my $fetch_tuple_sub = sub { $sth_slct->fetchrow_arrayref };
my @tuple_status;
$sth_ins->execute_for_fetch($fetch_tuple_sub, \@tuple_status);
my @errors = grep { ref $_ } @tuple_status;
if (scalar(@errors) > 0)
{
    print("errors: " . join(", ", @errors));
}
$sth_slct->finish();
$sth_ins->finish();
$dbh->disconnect();
```

Получить количество обработанных записей (rows)

Назначение

Метод `rows` предоставляет количество записей, выбранных (модифицированных) предшествующим методом `execute`.

Пакет

Package Linter::st

Прототип

```
$rv = $sth->rows();
```

Возвращаемые значения

Переменная	Описание
`$rv`:	
количество обработанных записей	Нормальное завершение
<0	Ошибка обработки SQL-предложения
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Получить ссылку на массив значений следующей записи выборки данных (fetchrow_arrayref)

Назначение

Метод `fetchrow_arrayref` выбирает следующую запись предварительно выполненной выборки данных и возвращает ее как ссылку на массив значений этой записи.

Пакет

Package Linter::st

Прототип

```
$ary_ref = $sth->fetchrow_arrayref([\%attributes]);

%attributes = (
'direction' => {'first' | 'last' | 'next' | 'prev' | 'number'}
'number' => {1, 2, ...}
);
```

Параметр	Описание
'direction'	Задает направление перемещения в массиве ответов выполненного SQL-оператора (как правило, SELECT): • 'first' – получить первую запись; • 'last' – получить последнюю запись;

Параметр	Описание
	• 'next' – получить следующую запись; • 'prev' – получить предыдущую запись; • 'number' – получить запись с заданным номером

Возвращаемые значения

Переменная	Описание
`$ary_ref`:	
ссылка на массив данных, содержащий выбранную запись	Нормальное завершение
undef	Ошибка выполнения метода
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Примечание
Количество выбранных записей возвращают методы `execute` и `rows`.

Получить массив значений следующей записи выборки данных (fetchrow_array)

Назначение

Метод `fetchrow_array` выбирает следующую запись предварительно выполненной выборки данных и возвращает ее в виде массива.

Пакет

Package Linter::st

Прототип

```
@ary = $sth->fetchrow_array([\%attributes]);

%attributes = (
'direction' => {'first' | 'last' | 'next' | 'prev' | 'number'}
'number' => {1, 2, ...}
);
```

Параметр	Описание
'direction'	Задает направление перемещения в массиве ответов выполненного SQL-оператора (как правило, SELECT): • 'first' – получить первую запись; • 'last' – получить последнюю запись; • 'next' – получить следующую запись; • 'prev' – получить предыдущую запись;

Параметр	Описание
	• 'number' – получить запись с заданным номером

Возвращаемые значения

Переменная	Описание
`@ary`:	
массив или список данных, содержащий выбранную запись	Нормальное завершение
undef	Ошибка выполнения метода
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Примечание
Количество выбранных записей возвращают методы `execute` и `rows`.

Примеры

```
1)
# выборка из таблицы БД записей в виде списка
while (($name, $tip, $tip1) = $sth->fetchrow_array)
{
    print "$name $tip1 $tip2";
}

2)
# выборка из таблицы БД записей в виде массива
while (@row = $sth->fetchrow_array)
{
    print "$row[0] $row[1] $row[2]";
}
```

В обоих примерах цикл while выполняется до тех пор, пока метод `fetchrow_array` не вернёт false.

Получить ссылку на хеш-массив следующей записи выборки данных (fetchrow_hashref)

Назначение

Метод `fetchrow_hashref` выбирает следующую запись предварительно выполненной выборки данных и возвращает ее как ссылку на хеш-массив, содержащий ключ в виде имени поля выборки данных и значение в виде значения поля выборки данных. NULL-значения полей выборки данных в хеш-массиве представлены undef-значениями.

Если выборка данных содержит дубликаты имен полей, то в выходном хеш-массиве будет представлен только один такой хеш-ключ. Например, для запроса

```
select MODEL, MAKE, COLOR, MODEL from AUTO;
```

метод fetchrow_hashref вернет только один хеш-ключ для поля MODEL. В такой ситуации необходимо использовать псевдонимы столбцов или метод fetchrow_arrayref.

Примечание
В связи с дополнительными затратами на формирование хеш-массива метод fetchrow_hashref менее эффективен по сравнению с методами fetchrow_arrayref или fetchrow_array.

По умолчанию ссылка на новый хеш-массив возвращается для каждой записи выборки данных, даже если записи полностью идентичны.

Пакет

Package Linter::st

Прототип

```
$hash_ref = $sth->fetchrow_hashref([\%attr]);
%attr = ('direction' => {'first' | 'last' | 'next' | 'prev' |
 'number'}
         'number'    => {1, 2, ...}
);
```

Возвращаемые значения

Переменная	Описание
$hash_ref	Ссылка на хеш-массив текущей записи выборки данных, undef при достижении конца выборки данных или возникновении ошибки
%attr	Ссылка на хеш-массив атрибутов для задания дополнительных инструкций по перемещению по выборке данных. Значения атрибута direction: • 'first' – получить первую запись; • 'last' – получить последнюю запись; • 'next' – получить следующую запись; • 'prev' – получить предыдущую запись; • 'number' – получить запись с заданным номером. Атрибут number используется совместно только при задании в атрибуте direction значения number и указывает номер записи в выборке, которую необходимо получить
$DBI::errstr	Диагностическое сообщение (в случае ошибки)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
or die "Could not connect to database: " . DBI->errstr;
my $sth = $dbh->prepare("select MAKE, MODEL, YEAR from AUTO;");
$sth->execute();
```

```
while (my $data_ref = $sth->fetchrow_hashref)
{
    print("MAKE = $data_ref->{MAKE}\n");
    print("MODEL = $data_ref->{MODEL}\n");
    print("YEAR = $data_ref->{YEAR}\n");
}
$sth->finish();
$dbh->disconnect();
```

Получить массив ссылок на записи выборки данных выполненного запроса (fetchall_arrayref)

Назначение

Метод `fetchall_arrayref` предоставляет ссылку на массив всей выборки данных предварительно выполненного SQL-запроса, где каждая запись выборки данных представлена ссылкой на массив значений полей SQL-запроса.

Если выборка данных пуста, метод возвращает ссылку на пустой массив. Если в процессе выполнения метода по какой-либо причине (например, разрыв соединения) драйвер больше не сможет предоставлять записи выборки данных, то возвращаемый массив будет содержать ссылки только на ранее выбранные записи. Необходимо контролировать подобные ситуации, например, путем вызова $sth->err для уточнения возможного усечения выборки данных из-за возникшей ошибки.

Параметр `$slice` задает номера столбцов выборки данных для формирования записей, на которые будут создаваться ссылки в возвращаемом массиве. Если параметр не определён, то в возвращаемом массиве будут ссылки на записи, содержащие значения всех столбцов выборки данных. Нумерация элементов в массиве начинается с `0`.

Если параметр `$slice` является ссылкой на хеш-массив, содержащий хеш-ключи (имена) столбцов, то метод `fetchall_arrayref` формирует записи по хеш-ключам этого массива. Если хеш-массив не пуст, то он используется для формирования записей, на которые создаются ссылки в возвращаемом массиве ссылок, по именам столбцов. Значения хеш-массива должны быть установлены в `1`.

Предоставлять значения всех столбцов выборки данных в виде ссылки на массив значений полей выборки:

```
$tbl_ary_ref = $sth->fetchall_arrayref([]);
```

Предоставлять значения только первого столбца каждой записи:

```
$tbl_ary_ref = $sth->fetchall_arrayref([0]);
```

Предоставлять значения всех столбцов выборки данных в виде ссылки на хеш-массив:

```
$tbl_ary_ref = $sth->fetchall_arrayref({});
```

Предоставлять значения столбцов MODEL и MAKE в виде ссылки на хеш-массив:

```
$tbl_ary_ref = $sth->fetchall_arrayref({ MODEL => 1, MAKE => 1 });
```

Если `$slice` является ссылкой на хеш-массив, то хеш-значения этого массива можно использовать для указания выбираемых столбцов и их переименования. Номера

выбираемых столбцов задаются порядковым номером хеш-значения в хеш-массиве (отсчет начинается с 0) и, при необходимости, указанием нового имени столбца.

Например, выборка значений только первого и второго столбца (MODEL, MAKE) таблицы AUTO и присвоения им нового имени CAR_MODEL, CAR_MAKE

```
$tbl_ary_ref = $sth->fetchall_arrayref(\{ 0=>'CAR_MODEL',
 1=>'CAR_MAKE' });
```

Если параметр $max_rows определен и имеет значение больше или равное нулю, то он используется для ограничения количества извлекаемых записей выборки данных.

Пакет

Package Linter::st

Прототип

```
$tbl_ary_ref = $sth->fetchall_arrayref;

$tbl_ary_ref = $sth->fetchall_arrayref($slice);

$tbl_ary_ref = $sth->fetchall_arrayref($slice, $max_rows);
```

Параметр	Описание
$slice	Контролирует формат возвращаемых записей массива и может задавать столбцы, которые должны быть включены в выходной массив (см. описание метода)
$max_rows	Максимальное количество выбираемых записей выборки данных

Возвращаемые значения

Переменная	Описание
$tbl_ary_ref	Ссылка на массив ссылок отобранных записей выборки данных. Значение undef возвращается при выполнении метода по неактивному объекту Statement (запрос не выполнен). Пустой массив возвращается при отсутствии данных
$DBI::errstr	Диагностическое сообщение (в случае ошибки)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
or die "Could not connect to database: " . DBI->errstr;
my $sth = $dbh->prepare("select MODEL, MODEL, WEIGHT, COLOR from
 AUTO;");
$sth->execute();
my $ary_ref = $sth->fetchall_arrayref([0, 3], 5);
for (my $i = 0; $i < scalar(@$ary_ref); $i++)
{
    for (my $j = 0; $j < scalar(@{$ary_ref->[$i]}); $j++)
    {
        print($ary_ref->[$i][$j] . "\n");
```



```
    }
    print("\n");
}
$sth->finish();
$dbh->disconnect();
```

Получить массив ссылок на хеш-массив значений столбца всех записей выборки данных выполненного запроса (fetchall_hashref)

Назначение

Метод `fetchall_hashref` предоставляет ссылку на хеш-массив всех значений указанного столбца выборки данных из предварительно выполненного SQL-запроса. Он возвращает ссылку на хеш-массив, содержащий хеш-ключ для каждого уникального значения указанного столбца выборки данных. Для каждого хеш-ключа соответствующее значение является ссылкой на хеш-массив, содержащий все выбранные столбцы и их значения.

Если выборка данных пуста, метод `fetchall_hashref` вернет ссылку на пустой хеш-массив. Если в процессе выполнения метода по какой-либо причине (например, разрыв соединения) драйвер больше не сможет предоставлять записи выборки данных, то возвращаемый хеш-массив будет содержать ссылки только на ранее выбранные записи. Необходимо контролировать подобные ситуации, например, путем вызова $sth->err для уточнения возможного усечения выборки данных из-за возникшей ошибки.

Параметр `$key_field` содержит имя поля, которое будет использоваться как хеш-ключ для полученных хеш-значений. Параметр также может содержать значение в виде порядкового номера столбца выборки данных (отсчет начинается с `1`). Если параметр `$key_field` задан неправильно (в выборке данных нет столбцов с таким номером или именем), то возвращается ошибка. Для запросов, возвращающих более одного «ключевого» столбца, можно указать несколько имен столбцов, передав `$key_field` как ссылку на массив, содержащий одно или несколько ключевых имен столбцов (или индексов), например:

```
$sth = $dbh->prepare("select YEAR, HORSEPWR, MAKE from AUTO;");
  $sth->execute();
  $hash_ref = $sth->fetchall_hashref([qw(YEAR HORSEPWR)]);
  print "Model '71 with 150 horsepower is $hash_ref->{71}->{150}-
>{MAKE}\n";
```

Пакет

Package Linter::st

Прототип

```
$hash_ref = $sth->fetchall_hashref($key_field);
```

Параметр	Описание
`$key_field`	Имя или порядковый номер столбца (столбцов) выборки данных, используемого для формирования его хеш-массива значений

Возвращаемые значения

Переменная	Описание
`$hash_ref`	Ссылка на массив ссылок хеш-массивов значений заданных столбцов выборки данных или ссылка на пустой хеш-массив в случае отсутствия данных
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
or die "Could not connect to database: " . DBI->errstr;
my $sth = $dbh->prepare("select SERIALNO, MODEL, MAKE, COLOR from
 AUTO;");
$sth->execute();
my $ary_ref = $sth->fetchall_hashref("SERIALNO");
my @keys = keys(%$ary_ref);
for (my $i = 0; $i < scalar(keys(%$ary_ref)); $i++)
{
    print($ary_ref->{$keys[$i]}->{MODEL} . "\n");
    print($ary_ref->{$keys[$i]}->{MAKE} . "\n");
    print($ary_ref->{$keys[$i]}->{COLOR} . "\n");
    print("\n");
}
$sth->finish();
$dbh->disconnect();
```

Получить первую запись выборки данных в виде массива значений полей записи (selectrow_array)

Назначение

Метод `selectrow_array` объединяет вызовы методов `prepare`, `execute` и `fetchrow_array` в один вызов. Если результатом SQL-запроса является непустая выборка данных, то метод возвращает первую запись выборки данных в виде массива элементов. Т.к. доступ к последующим записям сформированной выборки данных не поддерживается, то метод эффективно применять к SQL-запросам, возвращающим одну запись, например,

```
select model, make from auto where color = 'RED' limit 1;
select count(*) from person;
```

Пакет

Package Linter::db

Прототип

```
@row_ary = $dbh->selectrow_array($statement);

@row_ary = $dbh->selectrow_array($statement, \%attr);
```



```
@row_ary = $dbh->selectrow_array($statement, \%attr,
 @bind_values);
```

Параметр	Описание
`$statement`	Текст SQL-запроса (возможно, с параметрами). SQL-запрос может быть предварительно претранслирован (подготовлен с помощью метода `prepare`), тогда при выполнении метода `selectrow_array` выполнение метода `prepare` будет пропущено
`\%attr`	Значение параметра игнорируется
`@bind_values`	Массив, содержащий значения привязываемых параметров (если SQL-запрос содержит параметры)

Возвращаемые значения

Переменная	Описание
`@row_ary`	Задает имя, которое должно использовать как первая запись выборки данных в виде массива значений. В случае отсутствия данных или при наличии ошибок выполнения методов `prepare` и/или `execute` возвращается пустой массив
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки выполнения метода)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
my @bind_values = (71, 4);
my $sth = $dbh->prepare("select * from AUTO where YEAR = ? and
                         CYLNDERS = ?;");
my @row_ary = $dbh->selectrow_array($sth, undef, @bind_values);
for (my $i = 0; $i < scalar(@row_ary); $i++)
{
  print($row_ary[$i] . "\n");
}
$sth->finish();
$dbh->disconnect();
```

Получить первую запись выборки данных в виде ссылки на массив значений полей записи (selectrow_arrayref)

Назначение

Метод `selectrow_arrayref` объединяет вызовы методов `prepare`, `execute` и `fetchrow_arrayref` в один вызов. Если результатом SQL-запроса является непустая выборка данных, то метод возвращает первую запись выборки данных в виде ссылки на массив элементов. Т.к. доступ к последующим записям сформированной

выборки данных не поддерживается, то метод эффективно применять к SQL-запросам, возвращающим одну запись, например,

```
select model, make from auto where color = 'RED' limit 1;
select count(*) from person;
```

Пакет

Package Linter::db

Прототип

```
$ary_ref = $dbh->selectrow_arrayref($statement);

$ary_ref = $dbh->selectrow_arrayref($statement, \%attr);

$ary_ref = $dbh->selectrow_arrayref($statement, \%attr,
 @bind_values);
```

Параметр	Описание
`$statement`	Текст SQL-запроса (возможно, с параметрами). SQL-запрос может быть предварительно претранслирован (подготовлен с помощью метода `prepare`), тогда при выполнении метода `selectrow_arrayref` выполнение метода `prepare` будет пропущено
`\%attr`	Значение параметра игнорируется
`@bind_values`	Массив, содержащий значения привязываемых параметров (если SQL-запрос содержит параметры)

Возвращаемые значения

Переменная	Описание
`$ary_ref`	При наличии данных возвращается ссылка на массив, содержащий первую запись выборки данных. В случае отсутствия данных или при наличии ошибок выполнения методов `prepare` и/или `execute` возвращается undef
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки выполнения метода)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
my @bind_values = (71, 4);
my $sth = $dbh->prepare("select * from AUTO where YEAR = ? and
                         CYLNDERS = ?;");
my $ary_ref = $dbh->selectrow_arrayref($sth, undef, @bind_values);
for (my $i = 0; $i < scalar($@ary_ref); $i++)
{
    print($ary_ref->[$i] . "\n");
}
```

```
$sth->finish();
$dbh->disconnect();
```

Получить запись выборки данных в виде хеш-массива значений полей записи (selectrow_hashref)

Назначение

Метод `selectrow_hashref` объединяет вызовы методов `prepare`, `execute` и `fetchrow_hashref` в один вызов. Если результатом SQL-запроса является непустая выборка данных, то метод возвращает первую запись выборки данных в виде хеш-массива элементов.

Примечание
Хеш-массив (ассоциативный массив) – это массив, доступ к данным которого осуществляется при помощи ключа, ассоциированного со значением.

Т.к. доступ к последующим записям сформированной выборки данных не поддерживается, то метод эффективно применять к SQL-запросам, возвращающим одну запись, например,

```
select model, make from auto where color = 'RED' limit 1;
select count(*) from person;
```

Пакет

Package Linter::db

Прототип

```
$hash_ref = $dbh->selectrow_hashref($statement);

$hash_ref = $dbh->selectrow_hashref($statement, \%attr);

$hash_ref = $dbh->selectrow_hashref($statement, \%attr,
 @bind_values);
```

Параметр	Описание
`$statement`	Текст SQL-запроса (возможно, с параметрами). SQL-запрос может быть предварительно претранслирован (подготовлен с помощью метода `prepare`), тогда при выполнении метода `selectrow_hashref` выполнение метода `prepare` будет пропущено
`\%attr`	Значение параметра не используется, игнорируется
`@bind_values`	Массив, содержащий значения привязываемых параметров (если SQL-запрос содержит параметры)

Возвращаемые значения

Переменная	Описание
`$hash_ref`	При наличии данных возвращается ссылка на ассоциированный массив, содержащий первую запись выборки данных. В случае

Переменная	Описание
	отсутствия данных или при наличии ошибок выполнения методов `prepare` и/или `execute` возвращается undef
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки выполнения метода)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
my $statement = "select MODEL, COLOR, YEAR, CHKMILE,
                 (EXTRACT(YEAR from SYSDATE) - (1900 + YEAR)) as
 AGE,
                MAKE || ' (' || BODYTYPE || ')' from AUTO
                where CYLNDERS = 8;";
my $hash_ref = $dbh->selectrow_hashref($statement);
print($hash_ref->{MODEL} . "\n");   #MODEL
print($hash_ref->{COLOR} . "\n");   #COLOR
print($hash_ref->{YEAR} . "\n");    #YEAR
print($hash_ref->{CHKMILE} . "\n"); #CHKMILE
print($hash_ref->{AGE} . "\n");     #AGE
print($hash_ref->{""} . "\n");      #last unnamed column
$dbh->disconnect();
```

Получить значения всей выборки данных (selectall_array)

Назначение

Метод `selectall_array` объединяет вызовы методов `prepare`, `execute` и `fetchall_array` в один вызов. Если результатом SQL-запроса является непустая выборка данных, то метод возвращает всю выборку данных в виде массива записей, каждая из которых представлена массивом значений полей SQL-запроса.

Пакет

Package Linter::db

Прототип

```
@ary = $dbh->selectall_array($statement);

@ary = $dbh->selectall_array($statement, \%attr);

@ary = $dbh->selectall_array($statement, \%attr, @bind_values);
```

Параметр	Описание
`$statement`	Текст SQL-запроса (возможно, с параметрами). SQL-запрос может быть предварительно претранслирован (подготовлен с помощью метода `prepare`), тогда при выполнении метода

Параметр	Описание
	`selectall_array` выполнение метода `prepare` будет пропущено
`\%attr`	Ассоциированный массив атрибутов с дополнительными указаниями по обработке метода: 1) `MaxRow` – задает объем выборки данных (максимальное количество записей выборки); 2) `Slice` – контролирует формат возвращаемых записей массива и может задавать столбцы, которые должны быть включены в выходной массив. Например, при задании атрибута `{ Slice => {} }` каждая запись в выходной выборке данных будет представлена в виде ассоциативного массива, а при задании опции `{ Slice => [0] }` каждая запись выходной выборки данных будет содержать значения только первого столбца. • `{ Slice => [] }` – каждая запись массива является ссылкой на массив значений полей выборки данных; • `{ Slice => {} }` – каждая запись в выходной выборке данных будет представлена в виде ассоциативного массива (хеш-массива); • `{ Slice => [n1, …] }`, `{ Slice => {n1, …} }` – каждая запись в выходном массиве будет содержать значения только указанных номеров столбцов
`@bind_values`	Массив, содержащий значения привязываемых параметров (если SQL-запрос содержит параметры)

Возвращаемые значения

Переменная	Описание
`@ary`	При наличии данных возвращается массив записей, каждая из которых представлена массивом значений полей SQL-запроса. В случае отсутствия данных или при наличии ошибок возвращается пустой массив
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки выполнения метода)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
my @bind_values = (71, 4);
my $sth = $dbh->prepare("select * from AUTO where YEAR = ? and
                         CYLNDERS = ?;");
my @ary = $dbh->selectall_array($sth, undef, @bind_values);
for (my $i = 0; $i < scalar(@ary); $i++)
{
    for (my $j = 0; $j < scalar(@{$ary[$i]}); $j++)
    {
        print($ary[$i][$j] . "\n");
```

```
    }
    print("\n");
}
$sth->finish();
$dbh->disconnect();
```

Получить ссылку на массив значений всей выборки данных (selectall_arrayref)

Назначение

Метод `selectall_arrayref` объединяет вызовы методов `prepare`, `execute` и `fetchall_arrayref` в один вызов. Если результатом SQL-запроса является непустая выборка данных, то метод возвращает всю выборку данных в виде ссылки на массив записей, каждая из которых представлена ссылкой на массив значений полей SQL-запроса.

Пакет

Package Linter::db

Прототип

```
$ary_ref = $dbh->selectall_arrayref($statement);

$ary_ref = $dbh->selectall_arrayref($statement, \%attr);

$ary_ref = $dbh->selectall_arrayref($statement, \%attr,
 @bind_values);
```

Параметр	Описание
`$statement`	Текст SQL-запроса (возможно, с параметрами). SQL-запрос может быть предварительно претранслирован (подготовлен с помощью метода `prepare`), тогда при выполнении метода `selectall_arrayref` выполнение метода `prepare` будет пропущено
`\%attr`	См. метод selectall_array
`@bind_values`	Массив, содержащий значения привязываемых параметров (если SQL-запрос содержит параметры)

Возвращаемые значения

Переменная	Описание
`$ary_ref`	При наличии данных возвращается ссылка на массив записей, каждая из которых представлена ссылкой на массив значений полей SQL-запроса. В случае отсутствия данных или при наличии ошибок выполнения методов `prepare` и/или `execute` возвращается undef
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки выполнения метода)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
my $ary_ref = $dbh->selectall_arrayref("select * from AUTO;",
                                       { Slice => [0], MaxRows =>
 10 });
for (my $i = 0; $i < scalar(@$ary_ref); $i++)
{
  for (my $j = 0; $j < scalar(@{$ary_ref->[$i]}); $j++)
  {
    print($ary_ref->[$i][$j] . "\n");
  }
  print("\n");
}
$dbh->disconnect();
```

Получить значения всей выборки данных в виде массива хеш-массивов (selectall_hashref)

Назначение

Метод `selectall_hashref` объединяет вызовы методов `prepare`, `execute` и `fetchall_hashref` в один вызов. Если результатом SQL-запроса является непустая выборка данных, то метод возвращает ссылку на хеш-массив, элементами которого являются хеш-массивы значений полей SQL-запроса.

Пакет

Package Linter::db

Прототип

```
$hash_ref = $dbh->selectall_hashref($statement, $key_field);

$hash_ref = $dbh->selectall_hashref($statement, $key_field, \
%attr);

$hash_ref = $dbh->selectall_hashref($statement, $key_field, \
%attr, @bind_values);
```

Параметр	Описание
`$statement`	Текст SQL-запроса (возможно, с параметрами). SQL-запрос может быть предварительно претранслирован (подготовлен с помощью метода `prepare`), тогда при выполнении метода `selectall_hashref` выполнение метода `prepare` будет пропущено
`%key_field`	Задает имя, которое должно использовать как хеш-ключ возвращаемого хеш-массива. Параметр может быть задан в виде ссылки на массив с несколькими именами, что приведет к созданию дерева вложенных хешей. Если хеш-ключ добавляемой

Параметр	Описание
	в выборку данных записи совпадает с хеш-ключом ранее вставленной записей, то старая запись заменяется новой
\%attr	См. метод selectall_array
@bind_values	Массив, содержащий значения привязываемых параметров (если SQL-запрос содержит параметры)

Возвращаемые значения

Переменная	Описание
$hash_ref	При наличии данных возвращается ссылка на хеш-массив, элементами которого являются хеш-массивы значений полей SQL-запроса. В случае отсутствия данных или при наличии ошибок выполнения методов prepare и/или execute возвращается undef
$DBI::errstr	Диагностическое сообщение (в случае ошибки выполнения метода)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
      my $statement = "select MODEL, COLOR, YEAR, CHKMILE from
 AUTO;");
my $hash_ref = $dbh->selectall_hashref("select MODEL, COLOR, YEAR,
 CHKMILE
                                        from AUTO;", 2);
my @ary_keys = keys(%{ $hash_ref });
for (my $i = 0; $i < scalar(@ary_keys); $i++)
{
    my $key = $ary_keys[$i];
    print($hash_ref->{$key}{MODEL} . "\n");
    print($hash_ref->{$key}{COLOR} . "\n");
    print($hash_ref->{$key}{YEAR} . "\n");
    print($hash_ref->{$key}{CHKMILE} . "\n");
    print($hash_ref->{$key}{CHKMILE} . "\n");
    print("\n");
}
$dbh->disconnect();
```

Получить ссылку на массив значений первого столбца выборки данных (selectcol_arrayref)

Назначение

Метод selectcol_arrayref объединяет вызовы методов prepare, execute и выборку значений первого столбца из выборки данных в один вызов. Метод возвращает ссылку на массив, содержащий значения указанных столбцов выборки данных (по умолчанию первого столбца).

Пакет

Package Linter::db

Прототип

```
$ary_ref = $dbh->selectcol_arrayref($statement);

$ary_ref = $dbh->selectcol_arrayref($statement, \%attr);

$ary_ref = $dbh->selectcol_arrayref($statement, \%attr,
 @bind_values);
```

Параметр	Описание
`$statement`	Текст SQL-запроса (возможно, с параметрами). SQL-запрос может быть предварительно претранслирован (подготовлен с помощью метода `prepare`), тогда при выполнении метода `selectcol_arrayref` выполнение метода `prepare` будет пропущено
`\%attr`	Ассоциированный массив атрибутов с дополнительными указаниями по обработке метода: • `MaxRow`: задает объем выборки данных (максимальное количество записей выборки данных); • `Columns`: массив с порядковыми номерами столбцов в выборке данных, которые будут включены в выходной массив. Например, при задании атрибута `{ Column => [1, 2] }` в выходной выборке будут присутствовать значения первого и второго столбцов
`@bind_values`	Массив, содержащий значения привязываемых параметров (если SQL-запрос содержит параметры)

Возвращаемые значения

Переменная	Описание
`$ary_ref`	При наличии данных возвращается ссылка на массив значений указанных столбцов выборки данных (по умолчанию первого столбца). В случае отсутствия данных или при наличии ошибок выполнения методов `prepare` и/или `execute` возвращается undef
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки выполнения метода)

Пример

```
my $dbh = $drh->connect("DEMO", "SYSTEM", "MANAGER8")
        or die "Could not connect to database: " . DBI->errstr;
my $ary_ref = $dbh->selectcol_arrayref("select * from AUTO;",
                                       { Columns => [1, 2]});
for (my $i = 0; $i < scalar(@$ary_ref); $i++)
{
```

```
    print($ary_ref->[$i] . "\n");
}
$dbh->disconnect();
```

Преобразование ESC-последовательностей (native_sql)

Назначение

С помощью метода native_sql можно получить SQL-выражение с преобразованными ESC-последовательностями.

Пакет

Package Linter::db

Прототип

```
$rv = $dbh->native_sql($statement);
```

Параметр	Описание
$statement	Текст SQL-оператора с преобразованными ESC-последовательностями

Возвращаемые значения

Переменная	Описание
$rv:	
преобразованное SQL-выражение	Нормальное завершение
undef	Ошибка в процессе преобразования
$DBI::errstr	Диагностическое сообщение (в случае ошибки подготовки предложения)

Получить порцию BLOB-данных (blob_read)

Назначение

Метод blob_read позволяет получить порцию BLOB-данных.

Примечание
Метод применяется к таблице после того, как предложение подготовлено, выполнено и выбрана хотя бы одна запись, содержащая поле типа BLOB.

Пакет

Package Linter::st

Прототип

```
$blob = $sth->blob_read($field,$offset,$length[,\$blobref]);
```


Параметр	Описание
`$field`	Номер выбираемого BLOB-столбца (отсчет начинается с `1`)
`$offset`	Смещение в байтах порции данных в BLOB-столбце
`$length`	Длина порции данных (в байтах)
`$blobref`	Ссылка на буфер, в который должна быть помещена порция данных

Возвращаемые значения

Переменная	Описание
`$blob`:	
порция данных	Нормальное завершение
undef	Ошибка выполнения метода
`$blobref`	Ссылка на буфер с порцией данных
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Добавить порцию BLOB-данных (blob_write)

Назначение

Метод `blob_write` позволяет добавить порцию BLOB-данных.

Примечание
Метод применяется только после того, как предложение подготовлено, выполнено и выбрана хотя бы одна запись, содержащая поле типа BLOB.

Пакет

Package Linter::st

Прототип

```
$rc = $sth->func($field, $length, \$blob, 'blob_write');
```

Параметр	Описание
`$field`	Номер BLOB-столбца, к которому добавляются BLOB-данные (отсчет начинается с `1`)
`$length`	Длина добавляемой порции данных (в байтах)
`$blob`	Ссылка на буфер, в котором содержится добавляемая порция данных

Возвращаемые значения

Переменная	Описание
`$rc`:	
`1`	Нормальное завершение
`0`	Ошибка выполнения метода
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Удалить BLOB-данные (blob_delete)

Назначение

С помощью метода `blob_delete` осуществляется удаление BLOB-данных.

Примечание
Метод применяется только после того, как предложение подготовлено, выполнено и выбрана хотя бы одна запись, содержащая поле типа BLOB.

Пакет

Package Linter::st

Прототип

```
$rc = $sth->func([$field],'blob_delete');
```

Параметр	Описание
`$field`	Номер BLOB-столбца, из которого удаляются BLOB-данные (отсчет начинается с `1`)

Возвращаемые значения

Переменная	Описание
`$rc`:	
`1`	Нормальное завершение
`0`	Ошибка выполнения метода
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Завершить обработку SQL-оператора (finish)

Назначение

Метод `finish` позволяет закончить обработку SQL-оператора и освободить все ресурсы, выделенные методам `prepare`, `execute` и `fetch` для его выполнения.

Пакет

Package Linter::st

Прототип

```
$rc = $sth->finish();
```

Возвращаемые значения

Переменная	Описание
`$rc`:	
`1`	Нормальное завершение
`0`	Ошибка выполнения метода

Переменная	Описание
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Закрыть соединение с СУБД (disconnect)

Назначение

Метод `disconnect` закрывает ранее установленное с помощью метода `connect` или `connect_cached` соединение с СУБД.

Пакет

Package Linter::db

Прототип

```
$rc = $dbh->disconnect([$action]);
```

Параметр	Описание
`$action`	Действие при закрытии соединения: • `commit` – сохранить в БД все изменения последней транзакции по соединению $dbh; • `rollback` – отменить изменения в БД. По умолчанию выполняется `rollback`

Возвращаемые значения

Переменная	Описание
`$rc`:	
`1`	Нормальное завершение
`0`	Ошибка закрытия
`$DBI::errstr`	Диагностическое сообщение (в случае ошибки)

Останов ядра СУБД ЛИНТЕР (shut)

Назначение

Метод `shut` выполняет останов СУБД ЛИНТЕР.

Пакет

Package Linter::db

Прототип

```
$rc = $dbh->func($database, 'shut');
```

Параметр	Описание
`$database`	Имя узла локальной сети, на котором установлена СУБД ЛИНТЕР. Имя этого узла должно быть прописано в файле сетевой

Параметр	Описание
	конфигурации `nodetab` (см. документ «СУБД ЛИНТЕР. Сетевые средства»). Для останова ядра локальной СУБД ЛИНТЕР по умолчанию значение должно быть пустым

Возвращаемые значения

Переменная	Описание
`$rc`:	
`1`	Нормальное завершение
`0`	Ошибка выполнения метода

Приложение 1
Коды завершения LinPerl-интерфейса

Имя константы	Значение	Описание
LPERR_SUCCESS	0	Успешное завершение
LPERR_NOMEMORY	-1	Недостаточно памяти
LPERR_INVCURSOR	-2	Неверный идентификатор курсора/соединения
LPERR_INVCOLUMN	-3	Неверный номер столбца
LPERR_INVPARAMNUM	-4	Неверный номер параметра
LPERR_INVSTATE	-5	Неверное значение состояния курсора/соединения
LPERR_INVDIRECT	-6	Неверное направление перемещения по курсору
LPERR_UNKNOWNOPT	-7	Неизвестная опция курсора
LPERR_BADDTFORMAT	-8	Неверный формат данных типа «дата/время»
LPERR_NOBLOBCOL	-9	Отсутствие BLOB-столбца в курсоре
LPERR_NOPARENT	-10	Для курсора нет родительского соединения
LPERR_INVPARAMETER	-11	Функции было передано недопустимое число параметров
LPERR_LINERROR	-12	Внутренняя ошибка СУБД ЛИНТЕР
LPERR_NOTIMPLEMENTED	-999	Свойство не реализовано

Приложение 2
Типы данных СУБД ЛИНТЕР

Тип данных	Числовое значение	Описание
LDT_CHAR	1	Символьные строки фиксированной длины (максимальная длина 4000 байт)
LDT_INTEGER	2	Целые числа (4 байта)
LDT_REAL	3	Вещественные числа (4 байта)
LDT_DATE	4	Дата-время (16 байтов)
LDT_NUMERIC	5	Числа с фиксированной точкой (16 байтов)
LDT_BYTE	6	Байтовые строки фиксированной длины (максимальная длина 4000 байт)
LDT_BLOB	7	BLOB-данные (максимальная длина 2 Гбайта)
LDT_VARCHAR	8	Символьные строки переменной длины (максимальная длина 4000 байт)
LDT_VARBYTE	9	Байтовые строки переменной длины (максимальная длина 4000 байт)
LDT_BOOLEAN	10	Логическое значение (5 байт). Из Perl-программы BOOLEAN-значение видно как целое число со значениями 0/1 (False/True)
LDT_NCHAR	11	UNICODE-строки фиксированной длины (максимальная длина 2000 UNICODE-символа)
LDT_NVARCHAR	12	UNICODE-строки переменной длины (максимальная длина 2000 UNICODE-символа)
LDT_EXTFILE	1	Внешний файл. Из Perl-программы тип данных EXTFILE виден как строка длиной до 512 байтов. Чтобы привязать переменную типа EXTFILE, надо использовать функцию EXTFILE(). **Пример** `$dbh->do('create table extfile_test(i int, e extfile);');` `$sth=$dbh->prepare('insert into extfile_test values(?,EXTFILE(?));');`
LDT_BIGINT	2	Длинное целое (8 байт)
LDT_SMALLINT	2	Короткое целое (2 байта)
LDT_DOUBLE	3	Вещественное двойной точности (8 байт)

Типы данных BIGINT, INTEGER, SMALLINT имеют одинаковый числовой код. Фактический тип данных должен определяться с помощью длины возвращаемых данных.

Пример обработки типов данных в DBI-интерфейсе в приложении 4

Приложение 3
Коды завершения DBI-интерфейса

Имя константы	Значение	Описание
NOERR	0	Успешное завершение
NOTFOUND	100	Нет данных
ERR_NOCTH	-3050	Неверный номер контекста
ERR_NODBH	-3051	Неверный номер соединения
ERR_NOSH	-3052	Неверный номер предложения
ERR_NOTCURSOR	-3053	Канал не является курсором
ERR_NOTPREPARED	-3054	Предложение не подготовлено
ERR_INVIDX	-3055	Неверный номер привязываемого параметра
ERR_NOREF	-3056	Параметр не определен
ERR_INVTYP	-3057	Неверный тип параметра
ERR_INVVAL	-3058	Неверное значение параметра
ERR_EXEPROC	-3059	Хранимая процедура не может исполняться посредством метода `execute(),do()`
ERR_NOTMAINCH	-3060	Попытка выполнить запрос не по главному каналу
ERR_CORRUPT	-3061	Испорчено содержимое внутренних структур
ERR_NOMEM	-3062	Нет памяти
ERR_SPNOTSUPP	-3063	ЛИНТЕР 4.xx не поддерживает вызовы хранимых процедур
ERR_NOCOMPAT	-3064	Несовместимые версии библиотеки и сервера
ERR_NONSELECT	-3065	Невозможно осуществить выборку данных в результате не SELECT-запроса

Приложение 4
Пример обработки типов данных в DBI-интерфейсе

```
#!/usr/bin/perl
require DBI;
import DBI;

$t->{'t'} = "t";

$drh = DBI->install_driver('Linter');

print "Version = $drh->{Version}";

$database='';
$username='SYSTEM';
$auth='MANAGER8';
$dbh = DBI->connect($database, $username, $auth, \%attr,
 'Linter');

if(defined $dbh){
 $dbh->do('drop table bigint_test;');
 $dbh->do('create table bigint_test(b boolean, i int, j int, bi
 bigint, ch varchar(15), ch1 char(20));');
 $dbh->do('insert into bigint_test values(true, 10, 10,
 42949672950, \'ddd\',\'\');');
 $dbh->do('insert into bigint_test values(false, 20, 10,
 429496729500, \'ddd\',\'\');');
 $dbh->do('insert into bigint_test values(true, 30, 10,
 -42949672950, \'ddd\',\'\');');
 $dbh->do('insert into bigint_test values(false, 40, 10,
 -429496729500, \'ddd\',\'\');');


$sth=$dbh->prepare('insert into bigint_test
 values(?,?,?,?,?,?);');
if(defined $sth){
# @param=(100, 100, "-42949672950", "fff, "ff");
 @param=(1, 100, 100, -42949672950000, "fff", "ff");
 $sth->execute(@param);
 $sth->execute(@param);
 $sth->execute(@param);
}

$sth->finish;
$sth=$dbh->prepare('select b,i,bi,j,ch,ch1 from bigint_test;');
```

```
$t = $sth->{TYPE};

print  "\n$t->[0], $t->[1], $t->[2], $t->[3], $t->[4], $t->[5]\n";

#die "\n";

$rv = $sth->execute();
while(1){
 @row=$sth->fetchrow_array;
 last unless defined $row[0];
 print "\n-> $row[0] $row[1] $row[2] $row[3] $row[4] $row[5]<-\n";
 print $row[1] /100, "\n";
};

}else{
 die("\nconnection error\n");
}

die "\n";
```


Указатель функций LinPerl-интерфейса

Указатель методов DBI-интерфейса