СИСТЕМА
УПРАВЛЕНИЯ
БАЗАМИ
ДАННЫХ

ЛИНТЕР®

ЛИНТЕР БАСТИОН

ЛИНТЕР СТАНДАРТ

# Ruby-интерфейс

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

Правообладателем продуктов ЛИНТЕР® является компания РЕЛЭКС (1990-2026). Все права защищены.

Данный документ является результатом интеллектуальной деятельности, права на который принадлежат компании РЕЛЭКС.

Все материалы данного документа, а также его части/разделы могут свободно размещаться на любых сетевых ресурсах при условии указания на них источника документа и активных ссылок на сайты компании РЕЛЭКС: relex.ru и linter.ru.

При использовании любого материала из данного документа несетевым/печатным изданием обязательно указание в этом издании источника материала и ссылок на сайты компании РЕЛЭКС: relex.ru и linter.ru.

Цитирование информации из данного документа в средствах массовой информации допускается при обязательном упоминании первоисточника информации и компании РЕЛЭКС.

Любое использование в коммерческих целях информации из данного документа, включая (но не ограничиваясь этим) воспроизведение, передачу, преобразование, сохранение в системе поиска информации, перевод на другой (в том числе компьютерный) язык в какой-либо форме, какими-либо средствами, электронными, механическими, магнитными, оптическими, химическими, ручными или иными, запрещено без предварительного письменного разрешения компании РЕЛЭКС.

## 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: info@linter.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** ..... 2
Назначение документа ..... 2
Для кого предназначен документ ..... 2
Необходимые предварительные знания ..... 2
Дополнительные документы ..... 2
**Общие сведения о Ruby-интерфейсе** ..... 3
Необходимые условия применения ..... 3
**Сборка Ruby-интерфейса** ..... 4
Сборка Ruby-интерфейса для ОС Linux как разделяемой библиотеки ..... 4
**Методы Ruby-интерфейса** ..... 6
Общие сведения ..... 6
Управление доступом к БД ..... 7
Создать объект Connection ..... 7
Установить соединение ..... 7
Установить опцию соединения (курсора) ..... 9
Получить статус соединения (курсора) ..... 10
Закрыть соединение ..... 11
Освободить соединение ..... 11
Открыть курсор ..... 12
Закрыть курсор ..... 13
Получение метаданных БД ..... 13
Получить описание параметров БД ..... 13
Получить информацию о табличных объектах БД ..... 14
Получить информацию о столбцах таблицы БД ..... 16
Обработка SQL-запросов ..... 18
Преобразование ESC-последовательности ..... 18
Выполнить подготовленный запрос ..... 19
Транслировать запрос ..... 20
Получить описание параметров претранслированного запроса ..... 21
Подставить параметры в претранслированный запрос ..... 22
Выполнить претранслированный запрос ..... 23
Обработка результата выполнения запроса ..... 24
Получить количество обработанных записей ..... 24
Получить значение выходных параметров хранимой процедуры ..... 25
Получить заданную запись выборки данных в виде массива значений ..... 26
Получить заданную запись выборки данных в виде ассоциированного массива ..... 27
Получить порцию записей выборки данных в виде массива значений ..... 29
Получить порцию записей выборки данных в виде ассоциированного массива ..... 30
Получить информацию о записи выборки данных ..... 31
Получить последний ROWID в заданном соединении (курсоре) ..... 33
Получить последний AUTOINC в заданном соединении (курсоре) ..... 34
Обработка BLOB-данных ..... 34
Общие сведения ..... 34
Получить порцию BLOB-данных ..... 35
Добавить порцию BLOB-данных ..... 36
Очистить BLOB-данные ..... 37
**Коды завершения Ruby-интерфейса** ..... 39
**Указатель методов** ..... 40

# Предисловие

## Назначение документа

Документ содержит описание Ruby-интерфейса с СУБД ЛИНТЕР.

Документ предназначен для СУБД ЛИНТЕР БАСТИОН 6.0 сборка 20.8, далее по тексту СУБД ЛИНТЕР.

## Для кого предназначен документ

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

## Необходимые предварительные знания

Для работы необходимо владеть:

- основами реляционных баз данных и языка баз данных SQL;
- языком программирования Ruby;
- навыками работы в соответствующей операционной системе на уровне обычного пользователя.

## Дополнительные документы

- Справочник кодов завершения
- Интерфейс нижнего уровня
- Справочник по SQL
- ODBC-драйвер


© Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

# Общие сведения о Ruby-интерфейсе

Ruby (англ. «Рубин») – интерпретируемый язык высокого уровня для быстрого и удобного объектно-ориентированного программирования. Язык обладает независимой от операционной системы реализацией многопоточности, строгой динамической типизацией, сборщиком мусора и многими другими возможностями. Ruby-интерфейс предназначен для доступа к СУБД ЛИНТЕР приложений, разработанных на языке Ruby.

# Необходимые условия применения

Для использования Ruby-интерфейса СУБД ЛИНТЕР на хост-компьютере должен быть установлен интерпретатор языка программирования Ruby.

Если интерпретатор Ruby не установлен, то его исходные тексты и инструкцию по установке можно найти на сайте https://www.ruby-lang.org.

Ruby-интерфейс поддерживает 1.8.x – 2.6.x версии Ruby.

**Примечание**
Если необходимая Вам версия отсутствует в перечне поддерживаемых версий, следует обратиться в раздел Поддержка на сайте ЛИНТЕР.

# Сборка Ruby-интерфейса

## Сборка Ruby-интерфейса для ОС Windows

Для сборки Ruby-интерфейса СУБД ЛИНТЕР в среде ОС Windows:

1) установить (если это не было сделано ранее) интерпретатор языка программирования Ruby в ОС;
2) установить пакет Ruby Development Tools для ОС Windows;
3) из командной строки Ruby Development Tools выполнить команды:

```
ruby extconf.rb
make
make install
```

После успешной компиляции и компоновки динамическая библиотека `LinRuby.so` будет размещена в специальном каталоге загрузки библиотек Ruby. Для просмотра списка каталогов для загрузки библиотек Ruby необходимо выполнить:

```
ruby -e 'puts $:'
```

4) перезапустить Web-сервер (если используется Ruby on Rails для WEB-приложения).

## Сборка Ruby-интерфейса для ОС Linux как разделяемой библиотеки

Для сборки библиотеки необходимо иметь:

- установленные заголовочные файлы Ruby;
- С-компилятор и набор утилит для сборки (make и т.п.).

Сборка библиотеки может быть осуществлена двумя способами:

- через специальный ruby-скрипт для сборки Ruby-расширения;
- через скрипт конфигурирования (configure) дистрибутива СУБД ЛИНТЕР.

Для сборки Ruby-интерфейса через ruby-скрипт необходимо:

1) перейти в подкаталог ruby установочного каталога СУБД;
2) выполнить команды:

```
ruby extconf.rb
make -f Makefile
make -f Makefile install
```

Для сборки Ruby-интерфейса через скрипт конфигурирования дистрибутива СУБД ЛИНТЕР необходимо:

1) запустить скрипт конфигурации configure из корневого каталога СУБД;
2) ответить утвердительно на вопрос о конфигурации дистрибутива для сборки Ruby-интерфейса;

3) определить местоположение заголовочных файлов Ruby, выбрав один из вариантов:
   - выполнить автоматический поиск заголовочных файлов Ruby;
   - указать вручную полный путь до заголовочных файлов Ruby. Если при вводе была допущена ошибка (по указанному пути файлы не найдены), будет предложено повторить ввод. Отказ от повторного ввода равносилен отказу от сборки Ruby-интерфейса;
   - установить значение переменной RUBY_INC в файле Definition дистрибутива СУБД ЛИНТЕР, задав полный путь до заголовочных файлов Ruby.
4) перейти в каталог /ruby дистрибутива СУБД ЛИНТЕР и выполнить команду:

```
make
```

для сборки интерфейса. После окончания компиляции и компоновки готовая к использованию разделяемая библиотека Ruby-интерфейс LinRuby.so будет находиться в подкаталоге /bin установочного каталога СУБД ЛИНТЕР;

5) скопировать разделяемую библиотеку LinRuby.so в специальный каталог загрузки библиотеки. Для просмотра списка каталогов для загрузки библиотек Ruby необходимо выполнить команду:

```
ruby -e 'puts $:'
```

# Методы Ruby-интерфейса

## Общие сведения

1) Взаимодействие клиентского приложения с ядром СУБД ЛИНТЕР осуществляется через два класса объектов: `Connection` и `Cursor`. Объект `Cursor` может быть открыт только на базе объекта Connection, так как наследует параметры, необходимые для работы с базой данных (БД).

   Объект `Connection` предоставляет следующие методы доступа к БД:

| Метод | Описание |
|---|---|
| `Create` | Создать объект `Connection` |
| `Open` | Открыть соединение с СУБД ЛИНТЕР |
| `Close` | Закрыть соединение |
| `Free` | Освободить соединение |
| `GetStatus` | Получить статус соединения |
| `GetDBInfo` | Получить описание параметров БД |
| `SQLExecuteDirect` | Выполнить подготовленный SQL-запрос |
| `GetRowCount` | Получить количество записей выборки данных |
| `SQLPrepare` | Транслировать SQL-запрос |
| `GetBindParamInfo` | Получить описание параметров, которые необходимо подставить в претранслированный SQL-запрос |
| `SQLBindParameter` | Подставить параметры в претранслированный SQL-запрос |
| `SQLExecute` | Выполнить претранслированный SQL-запрос |
| `GetProcOutParams` | Получить выходные параметры после выполнения хранимой процедуры |
| `SQLFetchRow` | Получить запись выборки данных в виде массива значений |
| `SQLFetchHash` | Получить запись выборки данных в виде ассоциированного массива |
| `SQLFetchManyRow` | Получить порцию записей выборки данных в виде массивов значений |
| `SQLFetchManyHash` | Получить порцию записей выборки данных в виде ассоциированных массивов |
| `SQLNativeSql` | Получить SQL-выражение с преобразованными ESC-последовательностями |
| `SQLTables` | Получить информацию о таблицах БД |
| `SQLColumns` | Получить информацию о столбцах таблицы |
| `GetAnswerInfo` | Получить информацию о структуре записи выборки данных или о конкретном столбце |
| `GetBlob` | Получить порцию BLOB-данных |
| `AddBlob` | Добавить порцию BLOB-данных |
| `DelBlob` | Очистить BLOB-данные |


© Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

| Метод | Описание |
|---|---|
| SetOption | Установить опцию соединения/курсора |
| GetLastRowId | Получить последний ROWID в заданном соединении/курсоре |
| GetLastAutoInc | Получить последний AUTOINC в заданном соединении/курсоре |

2) В случае ошибки выполнение метода прекращается.
3) С помощью функции (GetStatus) можно получить код ошибки.

   Возможные значения кодов завершения, возвращаемые методом GetStatus:

| Возвращаемое значение | Описание |
|---|---|
| 0 (NORMAL) | Нормальное завершение |
| Положительное | Код завершения СУБД ЛИНТЕР |
| Отрицательное | Код ошибки Ruby-модуля |

4) Причиной прекращения выполнения метода может быть как ошибка Ruby-модуля, так и результат обработки запроса к СУБД ЛИНТЕР. Если причиной является код завершения СУБД ЛИНТЕР, то методом GetStatus возвращается положительное значение. Все остальные (отрицательные значения) относятся к кодам завершения Ruby-модуля.
5) Коды завершения Ruby-модулей приведены в разделе «Коды завершения Ruby-интерфейса» , коды завершения СУБД ЛИНТЕР – в документе «Справочник кодов завершения».

# Управление доступом к БД

## Создать объект Connection

### Назначение

Метод Create создает объект Connection.

### Синтаксис

```
value LinterConnection.Create()
```

### Возвращаемое значение

Объект Connection.

## Установить соединение

### Назначение

Метод Open создает объект Connection, если объект еще не создан, и устанавливает соединение с СУБД ЛИНТЕР.

### Синтаксис

```
value {<объект Connect> | LinterConnection}.
Open (Name, Password[, Node[, Mode[, CharSet]]])
```

Конструкция `LinterConnection.Open(…)` создает объект `Connection` с заданными параметрами соединения и на его основе устанавливает соединение с СУБД ЛИНТЕР.

Конструкция `<объект Connection>.Open(…)` для соединения с СУБД ЛИНТЕР использует ранее созданный с помощью метода `Create()` объект `Connection`.

`Name`

Имя пользователя БД. Символьная строка длиной не более 66 символов.

`Password`

Пароль пользователя. Символьная строка длиной не более 18 символов.

`Node`

Имя ЛИНТЕР-сервера, с которым необходимо установить соединение. Символьная строка длиной не более 8 символов. Если параметр не задан (указано значение `nil`), то соединения осуществляется с сервером по умолчанию (локальным сервером).

`Mode`

Формат:

`[<режим транзакции>] [| <кодовая страница>]`

Возможные `<режимы транзакции>` соединения:

- `M_AUTOCOMMIT` – режим `AUTOCOMMIT`;
- `M_EXCLUSIVE` – режим `PESSIMISTIC`;
- `M_OPTIMISTIC` – режим `OPTIMISTIC`;

**Примечание**
Режим M_OPTIMISTIC устарел (использовать не рекомендуется).

Режим устанавливается как побитовая операция «|» между значением режима транзакции и кодовой страницей.

Значение `<кодовой страницы>` см. в описании аргумента `CharSet`.

Если `<режим транзакции>` не задан (указано значение `nil`), по умолчанию используется `M_AUTOCOMMIT`.

`CharSet`

Имя кодовой страницы соединения.

Список доступных кодовых страниц находится в системной таблице `LINTER_SYSTEM_USER.$$$CHARSET`.

Если кодовая страница не задана, содержит пустую строку или неверное имя кодовой страницы, то устанавливается кодовая страница по умолчанию (заданная через переменную окружения `LINTER_CP` или используемая по умолчанию интерфейсом нижнего уровня, см. документ «Интерфейс нижнего уровня»).

### Возвращаемое значение

Объект `Connection`.

 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

# Установить опцию соединения (курсора)

## Назначение

Метод `SetOption` устанавливает опцию объекта `Connection` или `Cursor`.

## Синтаксис

```
value {<объект Connect> | <объект Cursor>}.SetOption(Option,
 Value)
```

`Option`

Идентификатор устанавливаемой опции (таблица 1).

`Value`

Значение опции.

Таблица 1. Возможные значения параметра Option

| Опция | Значение опции | Описание |
|---|---|---|
| `FETCH_BLOB_MODE` | `true`\|`false` | Разрешает (`true`) или запрещает (`false`) вывод BLOB-данных в методах `SQLFetchRow`, `SQLFetchHash`, `SQLFetchManyRow`, `SQLFetchManyHash`. По умолчанию BLOB-данные выводятся. Опция применима к объектам `Connection`, `Cursor`. |
| `CHANNEL_PRIORITY` | Целочисленное значение в диапазоне 0-255 | Задает приоритет канала. По умолчанию приоритет канала равен нулю. Опция применима к объектам `Connection`, `Cursor`. |
| `DATE_FORMAT` | Символьный литерал | Задает формат представления значений типа «дата-время» в записях выборки данных. Если формат явно не установлен, то по умолчанию используется "DD-Mon-YYYY:HH24:MI:SS". Допустимые форматы описаны в документе «Справочник по SQL», функция `to_char()`. Опция применима к объектам `Connection`, `Cursor`. |
| `SET_SAVE_POINT` | Символьное значение | Устанавливает точку сохранения в текущей транзакции. Опция применима к объектам `Connection`, `Cursor`. |
| `SET_CURSOR_NAME` | Символьное значение (до 66 символов) | Задает имя курсора, используемого для команд позиционного обновления и удаления (`WHERE CURRENT OF`). Подробнее см. документы «Интерфейс |

| Опция | Значение опции | Описание |
|---|---|---|
| | | нижнего уровня» и «Справочник по SQL».<br>Опция применима только к объекту Cursor. |

### Возвращаемое значение

Объект Connection.

### Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  connect.SQLExecuteDirect("update PERSON set FIRSTNAM='Nicolas'
 where rowid = 1;")
  connect.SetOption(SET_SAVE_POINT, "1")
  puts "savepoint has set"
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

См. также пример demo29.rb в подкаталоге samples/ruby установочного каталога СУБД ЛИНТЕР.

## Получить статус соединения (курсора)

### Назначение

Метод GetStatus возвращает код выполнения последней операции.

### Синтаксис

```
value {<объект Connect> | <объект Cursor>}.GetStatus()
```

### Возвращаемое значение

| Возвращаемое значение | Описание |
|---|---|
| 0 (NORMAL) | Нормальное завершение |
| Положительное | Код завершения СУБД ЛИНТЕР |
| Отрицательное | Код ошибки Ruby-модуля |


© Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

### Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open()
  status = connect.GetStatus()
  puts "status = #{status}"
ensure
  if connect != nil
    connect.Close()
  end
end
```

## Закрыть соединение

### Назначение

Метод `Close` закрывает соединение с СУБД ЛИНТЕР.

Незавершенные транзакции (соединения и дочерних каналов) завершаются командой `commit`.

### Синтаксис

```
value <объект Connect>.Close()
```

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

### Пример

См. пример `demo1.rb` в подкаталоге `samples/ruby` установочного каталога СУБД ЛИНТЕР.

## Освободить соединение

### Назначение

Метод `Free` освобождает память, использовавшуюся при работе с соединением.

### Синтаксис

```
value <объект Connect>.Free()
```

**Примечание**
Если соединение является родителем одного или нескольких курсоров, то память, используемая этими курсорами, тоже будет освобождена.

См. пример `demo1.rb` в подкаталоге `samples/ruby` установочного каталога СУБД ЛИНТЕР.

# Открыть курсор

## Назначение

Метод `Open` открывает подчиненный канал на базе соединения, указанного как параметр, и создает объект «курсор», предназначенный для обработки SQL-запросов.

## Синтаксис

```
value LinterCursor.Open(Connection)
```

`Connection`

Ранее установленное соединение, на базе которого будет открыт объект `Cursor`.

## Возвращаемое значение

В случае удачного завершения возвращается объект `Cursor`.

## Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Open("SYSTEM", "MANAGER8", nil,
 M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  cursor = LinterCursor.Open(connect)
  puts "Cursor is successed"
  cursor.SetOption(SET_CURSOR_NAME, "CURSOR_1")
  puts "Cursor was named"
  cursor.SQLExecuteDirect("select NAME, JOB, CITY, AGE from
 PERSON;")
  puts "Before update with current of"
  hash = cursor.SQLFetchHash(FETCH_ABSNUM, 4)
  j = 0
  while ( j < hash.keys.size )
    puts "#{hash.keys[j]} = #{hash.values[j]}"
    j = j +1
  end
  cursor.SQLExecuteDirect("UPDATE PERSON SET AGE =  25 WHERE
 CURRENT OF CURSOR_1;")
  cursor.SQLExecuteDirect("select NAME, JOB, CITY, AGE, PERSONID
 from PERSON;") !=nil
  puts "After update with current of"
  hash = cursor.SQLFetchHash(FETCH_ABSNUM, 4)
  j = 0
  while ( j < hash.keys.size )
    puts "#{hash.keys[j]} = #{hash.values[j]}"
    j = j +1
  end
```

 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

```
rescue => error
  puts error
ensure
   if cursor != nil
    cursor.Close()
  end
  if connect != nil
    connect.Close()
  end
end
```

# Закрыть курсор

### Назначение

Метод `Close`, применённый к курсору, закрывает соответствующий этому курсору подчиненный канал соединения и освобождает выделенные для курсора ресурсы.

### Синтаксис

```
value <объект Cursor>.Close()
```

### Пример

См. пример `demo10.rb` в подкаталоге `samples/ruby` установочного каталога СУБД ЛИНТЕР.

# Получение метаданных БД

# Получить описание параметров БД

### Назначение

Метод `GetDBInfo` предоставляет информацию о параметрах БД.

### Синтаксис

```
value {<объект Connect> | <объект Cursor>}.GetDBInfo()
```

### Возвращаемое значение

Массив параметров БД.

Параметры БД возвращаются в виде ассоциированного массива, где ключи – имена параметров, а поля – значения этих параметров:

| Параметр | Описание |
|---|---|
| `BaseName` | Имя БД |
| `Channel` | Номер канала (соединения) с БД |
| `DefCharSet` | Идентификатор кодовой страницы по умолчанию |
| `Flags` | Параметры установленного соединения |
| `Log` | Признак ведения файла-протокола |

| Параметр | Описание |
|---|---|
| MaxRecSize | Максимальная длина записи в таблице БД |
| Node | Узел сети ЛИНТЕР-сервера |
| Os | Идентификатор операционной системы сервера |
| Sync | Признак синхронизации ввода/вывода |
| SysLog | Признак активности системного журнала |
| VerBuild | Номер сборки версии СУБД ЛИНТЕР |
| VerMajor | Старший номер версии СУБД ЛИНТЕР, для которой построена БД |
| VerMinor | Младший номер версии СУБД ЛИНТЕР, для которой построена БД |
| UseCharSet | Идентификатор установленной кодовой страницы канала |
| UseCharSetName | Имя установленной кодовой страницы канала |

### Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Open("SYSTEM", "MANAGER8", nil,
 M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  bd_info = connect.GetDBInfo()
  if bd_info != nil
    i = 0
    while ( i < bd_info.keys.size )
      puts "#{bd_info.keys[i]} = #{bd_info.values[i]}"
      i = i + 1
    end
  else
    puts "Info is nil"
  end
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

См. также пример demo2.rb в подкаталоге samples/ruby установочного каталога СУБД ЛИНТЕР.

# Получить информацию о табличных объектах БД

### Назначение

Метод SQLTables предоставляет информацию о табличных объектах БД.

 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

### Синтаксис

```
value {<объект Connect> | <объект Cursor>}.SQLTables(UserName,
 TableName, TableType)
```

`UserName`

Шаблон имени владельцев объектов. Шаблон может содержать спецсимволы _ и % (см. документ «Справочник по SQL», конструкция «Предикат подобия»).

`TableName`

Шаблон имени искомых объектов.

`TableType`

Символьная строка, содержащая список типов искомых объектов (объекты поиска перечисляются через запятую):

- `TABLE` – пользовательская таблица;
- `SYSTEM TABLE` – системная таблица;
- `VIEW` – представление;
- `SYNONIM` – синоним таблиц.

### Возвращаемое значение

Объект `Connection` (`Cursor`). Результат можно получить с помощью функций `SQLFetchRow`, `SQLFetchHash`, `SQLFetchManyRow`, `SQLFetchManyHash`.

Структура возвращаемой записи:

| Столбец записи | Описание |
|---|---|
| `TABLE_TYPE` | Тип объекта, char(66) |
| `REMARKS` | Комментарий к объекту (при его наличии в таблице $$$COMMENTS БД) |
| `TABLE_CAT` | Пробелы, char(66) |
| `TABLE_SCHEM` | Схема объекта (имя владельца объекта), char(66) |
| `TABLE_NAME` | Имя объекта, char(66) |

### Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_866)
  puts "connection to RDBMS Linter is successed"
  connect.SQLTables('SYS%', '$$$%', 'VIEW, TABLE, SYSTEM TABLE')
  i = 1
  while ( i <= connect.GetRowCount )
    hash = connect.SQLFetchHash(FETCH_ABSNUM, i)
    j = 0
    while ( j < hash.keys.size )
      puts "#{hash.keys[j]} = #{hash.values[j]}"
      j = j +1
```

```
    end
    i = i + 1
  end
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

См. также пример `demo21.rb` в подкаталоге `samples/ruby` установочного каталога СУБД ЛИНТЕР.

# Получить информацию о столбцах таблицы БД

## Назначение

Метод `SQLColumns` предоставляет информацию о столбцах таблицы БД.

## Синтаксис

```
value {<объект Connect> | <объект Cursor>}.SQLColumns(UserName,
 TableName, ColumnName)
```

`UserName`

Шаблон имени владельца таблицы (может быть `nil`). Шаблон может содержать спецсимволы _ и % (см. документ «Справочник по SQL», конструкция «Предикат подобия»).

`TableName`

Шаблон имени таблицы. Если шаблон не задан (указано `nil`), информация предоставляется обо всех таблицах данного пользователя.

`ColumnName`

Шаблон имени столбца таблицы. Если шаблон не задан (указано `nil`), информация предоставляется обо всех столбцах таблицы (таблиц).

## Возвращаемое значение

Объект `Connection` (`Cursor`). Возвращаемые записи можно получить с помощью функций `SQLFetchRow`, `SQLFetchHash`, `SQLFetchManyRow`, `SQLFetchManyHash`.

Структура возвращаемой записи:

| Столбец записи | Описание |
|---|---|
| `TABLE_CAT` | Пробелы, char(66) |
| `TABLE_SCHEM` | Схема объекта (имя владельца таблицы), char(66) |
| `TABLE_NAME` | Имя таблицы, char(66) |
| `COLUMN_NAME` | Имя столбца, char(66) |
| `DATA_TYPE` | Числовой идентификатор типа данных столбца СУБД ЛИНТЕР: |

 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

| Столбец записи | Описание |
|---|---|
| | • 1 – 'character';<br>• 2 – 'smallint', 'integer', 'bigint' (действительный тип данных распознается по длине возвращаемого значения в поле CHAR_OCTET_LENGTH);<br>• 3 – 'real', 'double' (действительный тип данных распознается по длине возвращаемого значения в поле CHAR_OCTET_LENGTH);<br>• 4 – 'date';<br>• 5 – 'decimal';<br>• 6 – 'byte';<br>• 7 – 'blob';<br>• 8 – 'varchar';<br>• 9 – 'varbyte';<br>• 10 – 'boolean';<br>• 11 – 'nchar';<br>• 12 – 'nchar varying';<br>• 13 – 'extfile' |
| TYPE_NAME | Название типа данных (см. поле DATA_TYPE, символьный идентификатор типа данных) |
| COLUMN_SIZE | Возвращается для всех типов данных. Для типов данных с определяемой длиной (char, varchar, byte, varbyte, nchar, nvarchar) содержит максимальную длину значений столбца в символах |
| BUFFER_LENGTH | Длина буфера данных, выделяемого для SQLFetchRow |
| DECIMAL_DIGITS | Количество значимых чисел после точки |
| NUM_PREC_RADIX | Подробнее см. документ «ODBC-драйвер» |
| NULLABLE | Допускается или нет NULL-значение |
| REMARKS | Комментарий к столбцу |
| COLUMN_DEF | Значение столбца по умолчанию |
| SQL_DATA_TYPE | Числовой идентификатор типа данных столбца по стандарту ODBC |
| SQL_DATETIME_SUB | Код подтипа дат и промежуточных типов дат (по стандарту ODBC) |
| CHAR_OCTET_LENGTH | Максимальная длина столбца в байтах (для всех типов данных) |
| ORDINAL_POSITION | Порядковый номер столбца в таблице (нумерация начинается с 1) |
| IS_NULLABLE | Значением столбца является NULL-значение |

### Пример

```
require "LinRuby"
```

```
begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_866)
  puts "connection to RDBMS Linter is successed"
  connect.SQLColumns('SYSTEM', nil, nil)
  i = 1
  while ( i <= connect.GetRowCount )
    hash = connect.SQLFetchHash(FETCH_ABSNUM, i)
    j = 0
    while ( j < hash.keys.size )
      puts "#{hash.keys[j]} = #{hash.values[j]}"
      j = j +1
    end
    i = i + 1
  end
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

# Обработка SQL-запросов

## Преобразование ESC-последовательности

### Назначение

Метод `SQLNativeSql` выполняет преобразование ESC-последовательностей в SQL-выражении. Подробнее см. документ «ODBC-драйвер».

### Синтаксис

```
value {<объект Connect> | <объект Cursor>}.SQLNativeSql(Query)
```

`Query`

SQL-выражение.

### Возвращаемое значение

В случае удачного завершения функция возвращает преобразованную символьную строку.

### Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
```


© Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

```
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  Query = connect.SQLNativeSql("SELECT { fn CONVERT (DATA_TYPE,
 SQL_SMALLINT) } FROM TYPEINFO;")
  connect.SQLExecuteDirect(Query)
  i = 1
  while ( i <= connect.GetRowCount )
    hash = connect.SQLFetchHash(FETCH_ABSNUM, i)
    j = 0
    while ( j < hash.keys.size )
      puts "#{hash.keys[j]} = #{hash.values[j]}"
      j = j +1
    end
  i = i + 1
  end
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

# Выполнить подготовленный запрос

### Назначение

Метод `SQLExecuteDirect` выполняет подготовленный SQL-запрос.

### Синтаксис

```
value {<объект Connect> | <объект
 Cursor>}.SQLExecuteDirect(Query)
```

`Query`

SQL-предложение.

### Возвращаемое значение

Объект `Connect` (`Cursor`).

В случае выполнения хранимой процедуры результат и выходные параметры можно получить с помощью метода `GetProcOutParams`.

### Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
```

```
  puts "connection to RDBMS Linter is successed"
  connect.SQLExecuteDirect("select NAME, JOB, CITY from PERSON
 where FIRSTNAM='PHIL';")
  i = 1
  while ( i <= connect.GetRowCount )
    hash = connect.SQLFetchHash(FETCH_ABSNUM, i)
    j = 0
    while ( j < hash.keys.size )
      puts "#{hash.keys[j]} = #{hash.values[j]}"
      j = j +1
    end
    i = i + 1
  end
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

См. также пример `demo4.rb` в подкаталоге `samples/ruby` установочного каталога СУБД ЛИНТЕР.

# Транслировать запрос

## Назначение

Метод `SQLPrepare` транслирует SQL-предложение.

> **Примечание**
> Использование претранслированных SQL-предложений рекомендуется в случае многократного выполнения одного и того же запроса с разными значениями параметрами (см. документ «Справочник по SQL», раздел «SQL-операторы с параметрами»).

## Синтаксис

```
value {<объект Connect> | <объект Cursor>}.SQLPrepare(Query)

Query
```

SQL-выражение.

## Возвращаемое значение

Объект `Connect` (`Cursor`).

## Пример

```
require "LinRuby"

begin
```

 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

```
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  connect.SQLPrepare("insert into PERSON(NAME, FIRSTNAM, PERSONID)
 values(?,?,?);")
  connect.SQLBindParameter(1, 'Kity')
  connect.SQLBindParameter(2, 'Black')
  connect.SQLBindParameter(3, '12345')
  connect.SQLExecute()
  puts "insert of new record is successed"
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

См. также пример `demo9.rb` в подкаталоге `samples/ruby` установочного каталога СУБД ЛИНТЕР.

# Получить описание параметров претранслированного запроса

## Назначение

Метод `GetBindParamInfo` предоставляет описание параметров последнего претранслированного запроса в указанном соединении (курсоре).

## Синтаксис

```
value {<объект Connect> | <объект
 Cursor>}.GetBindParamInfo([Param])
```

`Param`

Порядковый номер или имя параметра претранслированного запроса.

Порядковый номер может использоваться для указания любого параметра, имя – только для именованного параметра.

Нумерация параметров начинается с 1.

Если аргумент `Param` не указан, то предоставляется описание всех параметров запроса.

## Возвращаемое значение

Описание параметров в виде массива из n-элементов, где n – число параметров претранслированного запроса (n равно 1 в случае указания конкретного параметра).

Структура элемента массива:

- тип данных параметра (в обозначении СУБД ЛИНТЕР);
- максимальная длина параметра в байтах;

- точность: количество знаков после запятой (только для вещественных значений, в противном случае 0);
- масштаб: (только для вещественных значений, в противном случае 0).

### Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  connect.SQLPrepare("insert into PERSON(PERSONID, NAME, FIRSTNAM)
 values(:N, :NAME, :FIRSTNAM);")
  bind_param_info = connect.GetBindParamInfo()
  i = 0
  while ( i < bind_param_info.size)
    j = 0
    while ( j < bind_param_info[i].size )
      puts "#{bind_param_info[i].keys[j]} =
 #{bind_param_info[i].values[j]}"
      j = j + 1
    end
    i = i + 1
  end
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

См. также примеры `demo6.rb`, `demo7.rb` в подкаталоге `samples/ruby` установочного каталога СУБД ЛИНТЕР.

# Подставить параметры в претранслированный запрос

### Назначение

Метод `SQLBindParameter` подставляет значение параметра в последний претранслированный по заданному соединению (курсору) запрос.

### Синтаксис

```
value {<объект Connect> | <объект Cursor>}.SQLBindParameter(Param,
 Value)

Param
```

Порядковый номер или имя параметра.

 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

Порядковый номер может использоваться для указания любого параметра, имя – только для именованного параметра.

Нумерация параметров начинается с 1.

`Value`

Значение параметра.

### Возвращаемое значение

Объект `Connect` (`Cursor`).

### Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  connect.SQLPrepare("insert into PERSON(NAME, FIRSTNAM, PERSONID)
 values(?,?,?);")
  connect.SQLBindParameter(1, 'Kity')
  connect.SQLBindParameter(2, 'Black')
  connect.SQLBindParameter(3, '12345')
  connect.SQLExecute()
  puts "insert of new record is successed"
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

См. также примеры `demo9.rb`, `demo10.rb` в подкаталоге `samples/ruby` установочного каталога СУБД ЛИНТЕР.

# Выполнить претранслированный запрос

### Назначение

Метод `SQLExecute` подставляет (при необходимости) параметры в последний претранслированный по заданному соединению (курсору) запрос и затем выполняет его.

### Синтаксис

```
value {<объект Connect> | <объект Cursor>}.SQLExecute ([Params])
```

`Params`

Массив значений параметров претранслированного запроса. Количество элементов массива значений параметров должно быть не меньше количества параметров претранслированного запроса (лишние значения параметров игнорируются).

### Возвращаемое значение

Объект `Connect` (`Cursor`).

В случае выполнения хранимой процедуры результат и выходные параметры можно получить с помощью функции `GetProcOutParams`.

### Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  connect.SQLPrepare("insert into PERSON(NAME, FIRSTNAM, PERSONID)
 values(?,?,?);")
  connect.SQLExecute(['Kity','Black', 1003])
  puts "insert of new record is successed"
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

См. также пример `demo12.rb` в подкаталоге `samples/ruby` установочного каталога СУБД ЛИНТЕР.

# Обработка результата выполнения запроса

## Получить количество обработанных записей

### Назначение

Метод `GetRowCount` возвращает число записей, обработанных последним предложением `SELECT`, `UPDATE`, `INSERT` или `DELETE` в указанном соединении (курсоре).

### Синтаксис

```
value {<объект Connect> | <объект Cursor>}.GetRowCount()
```

### Возвращаемое значение

Количество реально обработанных (найденных, модифицированных, добавленных или удаленных) записей.

### Пример

```
require "LinRuby"

begin
```

 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

```
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  connect.SQLExecuteDirect("select NAME, JOB, CITY from PERSON
 where FIRSTNAM='PHIL';")
  i = 1
  while ( i <= connect.GetRowCount )
    hash = connect.SQLFetchHash(FETCH_ABSNUM, i)
    j = 0
    while ( j < hash.keys.size )
      puts "#{hash.keys[j]} = #{hash.values[j]}"
      j = j +1
    end
    i = i + 1
  end
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

# Получить значение выходных параметров хранимой процедуры

### Назначение

Метод `GetProcOutParams` предоставляет результат и значение выходных параметров последней выполненной в указанном соединении (курсоре) хранимой процедуры.

### Синтаксис

```
value {<объект Connect> | <объект Cursor>}.GetProcOutParams()
```

### Возвращаемое значение

Массив значений.

Структура массива значений:

- нулевой элемент – результат выполнения хранимой процедуры (это может быть и курсор);
- первый и остальные элементы – значения выходных параметров (если процедура имеет выходные параметры).

### Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
```

```
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  connect.SQLExecuteDirect("EXECUTE TEST();")
  puts param = connect.GetProcOutParams()
  cursor = param[0]
  i = 1
  while ( i <= cursor.GetRowCount )
    hash = cursor.SQLFetchHash(FETCH_ABSNUM, i)
    j = 0
    while ( j < hash.keys.size )
      puts "#{hash.keys[j]} = #{hash.values[j]}"
      j = j +1
    end
    i = i + 1
  end
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

См. также примеры `demo10.rb` и `demo12.rb` в подкаталоге `samples/ruby` установочного каталога СУБД ЛИНТЕР.

# Получить заданную запись выборки данных в виде массива значений

## Назначение

Метод `SQLFetchRow` предоставляет заданную запись выборки данных последнего поискового запроса в указанном соединении (курсоре).

## Синтаксис

```
value {<объект Connect> | <объект Cursor>}.SQLFetchRow(Pos[,
 Number])

Pos
```

Целочисленный идентификатор местоположения записи в выборке данных:

| Аргумент Pos | Значение | Описание |
|---|---|---|
| 1 | `FETCH_FIRST` | Выбрать первую запись |
| 2 | `FETCH_LAST` | Выбрать последнюю запись |
| 3 | `FETCH_NEXT` | Выбрать следующую запись |
| 4 | `FETCH_PREV` | Выбрать предыдущую запись |
| 5 | `FETCH_ABSNUM` | Выбрать запись по абсолютному номеру |

 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

`Number`

Номер записи.

Для идентификатора местоположения записи `FETCH_FIRST`, `FETCH_LAST`, `FETCH_NEXT`, `FETCH_PREV` номер записи игнорируется, а для `FETCH_ABSNUM` должен быть указан обязательно.

Нумерация записей начинается с 1.

### Возвращаемое значение

Массив записей выборки данных.

### Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  connect.SQLExecuteDirect("select NAME, JOB, CITY, AGE, PERSONID
 from PERSON;")
  i = 1
  while ( i <= connect.GetRowCount )
    row = connect.SQLFetchRow(FETCH_ABSNUM, i)
    j = 0
    while ( j < row.size )
      puts row[j]
      j = j + 1
    end
    i = i + 1
  end
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

## Получить заданную запись выборки данных в виде ассоциированного массива

### Назначение

Метод `SQLFetchHash` предоставляет заданную запись выборки данных последнего поискового запроса в указанном соединении (курсоре) в виде ассоциированного массива, то есть в виде массива записей, где каждый элемент записи выборки данных представлен в виде пары значений:

```
<имя поля записи> <значение поля записи>
```

### Синтаксис

```
value {<объект Connect> | <объект Cursor>}.SQLFetchHash(Pos[,
 Number])
```

`Pos`

Целочисленный идентификатор местоположения записи в выборке данных (см. метод `SQLFetchRow`).

`Number`

Номер записи.

Для идентификатора местоположения записи `FETCH_FIRST`, `FETCH_LAST`, `FETCH_NEXT`, `FETCH_PREV` номер записи игнорируется, а для `FETCH_ABSNUM` должен быть указан обязательно.

Нумерация записей начинается с 1.

### Возвращаемое значение

Ассоциированный массив записей выборки данных.

В ассоциированном массиве каждому столбцу записи выборки данных соответствуют два элемента:

```
<имя столбца> <значение столбца в данной строке ответа>
```

### Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  connect.SQLExecuteDirect("select NAME, JOB, CITY, AGE, PERSONID
 from PERSON;")
  i = 1
  while ( i <= connect.GetRowCount )
    hash = connect.SQLFetchHash(FETCH_ABSNUM, i)
    j = 0
    while ( j < hash.values.size )
      puts "#{hash.keys[j]} = #{hash.values[j]}"
      j = j + 1
    end
    i = i + 1
  end
rescue => error
  puts error
```

 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

```
ensure
  if connect != nil
    connect.Close()
  end
end
```

# Получить порцию записей выборки данных в виде массива значений

## Назначение

Метод SQLFetchManyRow предоставляет заданное количество записей текущей выборки данных указанного соединения или курсора, начиная с заданного местоположения в выборке. Записи представлены в виде массива значений.

## Синтаксис

```
value {<объект Connect> | <объект Cursor>}.SQLFetchManyRow(Count[,
 Number])
```

Count

Размер порции (количество запрашиваемых записей).

Number

Начало порции (номер записи в выборке данных). Если аргумент Number не задан, порция начинается с первой записи выборки.

Нумерация записей начинается с 1.

## Возвращаемое значение

Массив записей выборки данных. Размерность массива соответствует реальному количеству выбранных записей (количество записей может оказаться меньше затребованной порции, если при заполнении порции был достигнут конец выборки данных).

## Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  connect.SQLExecuteDirect("select NAME, JOB, CITY, AGE, PERSONID
 from PERSON;")
  row = connect.SQLFetchManyRow(10)
  j = 0
  while ( j < row.size )
    p row[j]
```

```
    j = j + 1
  end
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

См. также пример `demo16.rb` в подкаталоге `samples/ruby` установочного каталога СУБД ЛИНТЕР.

# Получить порцию записей выборки данных в виде ассоциированного массива

## Назначение

Метод `SQLFetchManyHash` предоставляет заданное количество записей текущей выборки данных указанного соединения (курсора), начиная с заданного местоположения в выборке. Записи представлены в виде ассоциированного массива.

## Синтаксис

```
value {<объект Connect> | <объект Cursor>}.SQLFetchManyHash
 (Count[, Number])
```

`Count`

Размер порции (количество запрашиваемых записей).

`Number`

Начало порции (номер записи выборки). Если аргумент `Number` не задан, порция начинается с первой записи выборки.

Нумерация записей начинается с 1.

## Возвращаемое значение

Ассоциированный массив записей выборки данных. Размерность массива соответствует реальному количеству выбранных записей.

## Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  connect.SQLExecuteDirect("select NAME, JOB, CITY, AGE, PERSONID
 from PERSON;")
```

 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

```
  row = connect.SQLFetchManyHash(10)
  i = 0
  while ( i < row.size )
    hash = row[i]
    j = 0
    while ( j < hash.size )
      puts "#{hash.keys[j]} = #{hash.values[j]}"
      j = j + 1
    end
    i = i + 1
  end
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

См. также пример `demo18.rb` в подкаталоге `samples/ruby` установочного каталога СУБД ЛИНТЕР.

# Получить информацию о записи выборки данных

## Назначение

Метод `GetAnswerInfo` предоставляет информацию о структуре всей записи выборки данных или об отдельном её поле в виде ассоциированного массива.

## Синтаксис

```
value {<объект Connect> | <объект
 Cursor>}.GetAnswerInfo([Number])

Number
```

Порядковый номер столбца в записи выборки данных. Нумерация столбцов начинается с 1.

Если аргумент не задан, информация предоставляется обо всех столбцах записи выборки данных.

## Возвращаемое значение

Ассоциированный массив атрибутов столбца записи выборки данных (таблицы 2, 3).

В следующих случаях имена столбцов не предоставляются:

- если столбец в выборке имеет синоним (например, `select ... auto.model as "Модель"`) выдаются имя столбца или синоним;
- если значением столбца является выражение (например, функция `select sysdate;`) или выражение (`abs (x) + 67`);

- если значением столбца является объединение (пересечение, исключение) значений нескольких запросов:

Таблица 2. Структура описания простых столбцов

| Параметр | Описание |
|---|---|
| User | Имя владельца таблицы |
| Table | Имя таблицы |
| Column | Имя столбца |
| Length | Длина столбца |
| Type | Числовой идентификатор типа данных столбца (в спецификации СУБД ЛИНТЕР) |
| Precision | Точность (для числовых данных типа Numeric) |
| Scale | Масштаб (для данных типа Numeric) |
| CharSet | Идентификатор кодовой страницы |

Таблица 3. Структура описания BLOB-столбцов

| Параметр | Описание |
|---|---|
| User | Имя владельца таблицы |
| Table | Имя таблицы |
| Column | Имя столбца |
| Length | Длина столбца |
| Type | Тип данных столбца |
| Size | Размер BLOB-данных |
| TypeObj | Пользовательский тип BLOB-данных |
| Index_Time | Дата/время создания фразового индекса |

## Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  connect.SQLExecuteDirect("select NAME, JOB, CITY from PERSON
 where FIRSTNAM='PHIL';")
  i = 0
  answer_info = connect.GetAnswerInfo()
  if  answer_info != nil
    while ( i < answer_info.size )
      puts "#{answer_info.keys[i]}"
      j = 0
      while ( j < answer_info.values[i].size )
        puts "#{answer_info.values[i].keys[j]} =
 #{answer_info.values[i].values[j]}"
```

 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

```
        j = j + 1
      end
      i = i + 1
    end
  else
    puts "Answer info is nil"
  end
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

# Получить последний ROWID в заданном соединении (курсоре)

## Назначение

Метод GetLastRowId предоставляет значение последнего ROWID, который был обработан в заданном соединении (курсоре) при выполнении INSERT, UPDATE, DELETE запросов.

## Синтаксис

```
value {<объект Connect> | <объект Cursor>}.GetLastRowId()
```

## Возвращаемое значение

Последний обработанный ROWID.

## Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  connect.SQLExecuteDirect("insert into TR_DATA(C1_INT, C2_REAL)
 values (100, 2.5);")
  puts "Last RowId = #{connect.GetLastRowId()}"
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

# Получить последний AUTOINC в заданном соединении (курсоре)

### Назначение

Метод GetLastAutoInc предоставляет значение последнего AUTOINC, которое было добавлено в какую-либо базовую таблицу при выполнении последнего INSERT-запроса в заданном соединении (курсоре).

### Синтаксис

```
value {<объект Connect> | <объект Cursor>}.GetLastAutoInc()
```

### Возвращаемое значение

Последнее добавленное значение AUTOINC.

### Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_EXCLUSIVE | Q_ENCODE)
  puts "connection to RDBMS Linter is successed"
  connect.SQLExecuteDirect("insert into TR_DATA(C1_INT, C2_REAL)
 values (100, 2.5);")
  puts "Last AutoInc = #{connect.GetLastAutoInc()}"
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
  end
end
```

# Обработка BLOB-данных

## Общие сведения

Условия применения методов обработки BLOB-данных:

1) методы применяются к текущей строке выборки. Это означает, что предварительно должен быть выполнен SELECT-запрос и установлена текущая строка полученной выборки;
2) SELECT-запрос не должен содержать операций соединения (FROM tab1, tab2, ...) таблиц;
3) SELECT-запрос не должен содержать операций объединения (UNION), пересечения (INTERSECT) и т.п. других SELECT-запросов;
4) методы применимы к BLOB-столбцам, выбранным только из одной таблицы или обновляемого представления;

 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

5) методы не применимы к BLOB-столбцам, полученным из нескольких таблиц (обновляемых представлений).

# Получить порцию BLOB-данных

## Назначение

Метод `GetBlob` возвращает массив, содержащий BLOB-данные текущей записи последней выборки данных в заданном соединении (курсоре).

## Синтаксис

```
value {<объект Connect> | <объект Cursor>}.GetBlob([Start, Length,
 Number])
```

`Start`

Начало требуемой порции BLOB-данных (смещение порции данных задается в байтах, отсчет начинается с 1).

`Length`

Размер требуемой порции в байтах.

`Number`

Порядковый номер столбца в записи выборки данных. Нумерация начинается с 1.

Если метод вызывается без параметров, результат будет представлять собой массив, содержащий BLOB-данные всех столбцов максимальной доступной длины.

В случае, когда метод вызывается с одним параметром, результат будет представлять собой массив, содержащий BLOB-данные всех столбцов, при этом данные извлекаются с указанной позиции.

Если метод вызывается с двумя параметрам, результат будет представлять собой массив, содержащий BLOB-данные всех столбцов, извлеченные с указанной позиции (допустимо использование `nil`) и заданной длины.

Если метод вызывается с тремя параметрами, то результатом является массив, содержащий BLOB-данные указанного столбца, извлеченные с указанной позиции (допустимо использование `nil`) и заданной длины (допустимо использование `nil`).

## Возвращаемое значение

Массив, содержащий BLOB-данные одного или нескольких столбцов.

## Пример

```
require "LinRuby"

begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_AUTOCOMMIT)
  puts "connection to RDBMS Linter is successed"

  begin
  connect.SQLExecuteDirect("DROP TABLE testBlob;")
```

```
  rescue
  end

  connect.SQLExecuteDirect("CREATE TABLE testBlob(ID INT, B1
 BLOB);")
  puts "creating table testBlob is successed"
  blob_data = "BLOB_VALUE"

  connect.SQLExecuteDirect("insert into testBlob(B1) values
 (NULL);")
  connect.SQLExecuteDirect("SELECT B1 FROM testBlob;")
  connect.AddBlob(blob_data)
  puts "adding blob data is successed"

  connect.SQLExecuteDirect("SELECT * FROM testBlob;")

  value = connect.GetBlob(1)[0]

  if value != blob_data
    puts "error : result value of 'B1' blob column is #{value},
 expected #{ blob_data }\n"
  else
    puts "getting blob data is successed"
  end

  connect.SQLExecuteDirect("DROP TABLE testBlob;")

rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
    connect.Free()
  end
end
```

См. также пример в методе `AddBlob`.

# Добавить порцию BLOB-данных

## Назначение

Метод `AddBlob` добавляет порцию BLOB-данных в первый или указанный BLOB-столбец текущей записи последней выборки данных в заданном соединении (курсоре).

## Синтаксис

```
value {<объект Connect> | <объект Cursor>}.AddBlob(Value [,
 Number])
```

 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

`Value`

BLOB-данные.

`Number`

Порядковый номер BLOB-столбца в текущей записи выборки данных, к которому должна быть добавлена порция BLOB-данных. Если аргумент не задан по умолчанию данные добавляются в первый BLOB-столбец.

### Возвращаемое значение

Объект `Connection`.

### Пример

См. пример в методе `GetBlob`.

См. также пример `demo26.rb` в подкаталоге `samples/ruby` установочного каталога СУБД ЛИНТЕР.

## Очистить BLOB-данные

### Назначение

Метод `DelBlob` удаляет BLOB-данные из всех BLOB-столбцов или указанного BLOB-столбца текущей записи последней выборки данных в заданном соединении (курсоре).

### Синтаксис

```
value {<объект Connect> | <объект Cursor>}.DelBlob([Number])
```

`Number`

Порядковый номер BLOB-столбца в текущей записи выборки данных, из которого должны быть удалены BLOB-данные. Если аргумент не задан, по умолчанию удаляются данные из всех BLOB-столбцов.

### Возвращаемое значение

Объект `Connection`.

### Пример

```
begin
  connect = LinterConnection.Create()
  connect.Open("SYSTEM", "MANAGER8", nil, M_AUTOCOMMIT)
  puts "connection to RDBMS Linter is successed"

  connect.SQLExecuteDirect("CREATE TABLE testBlob(ID INT, B1
 BLOB);")

  blob_data = "BLOB_VALUE"
```

```
  connect.SQLPrepare("INSERT INTO testBlob(ID, B1) VALUES (?,?);")

  connect.SQLBindParameter(1, 1)
  connect.SQLBindParameter(2, blob_data)
  connect.SQLExecute()

  connect.SQLExecuteDirect("SELECT B1 FROM testBlob;")

  connect.DelBlob()

  connect.SQLExecuteDirect("SELECT B1 FROM testBlob;")

  value = connect.GetBlob(1)[0]
  expected_value = ""

  if value != expected_value
    puts "error : result value of 'B1' blob column is #{value},
 expected  #{expected_value}\n"
  end

  connect.SQLExecuteDirect("DROP TABLE testBlob;")
rescue => error
  puts error
ensure
  if connect != nil
    connect.Close()
    connect.Free()
  end
end
```

 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026

# Коды завершения Ruby-интерфейса

Коды завершения Ruby-интерфейса в таблице 4.

Таблица 4. Коды завершения Ruby-интерфейса

| Имя константы | Значение | Описание |
|---|---|---|
| NORMAL | 0 | Успешное завершение |
| NO_MEMORY | -1 | Недостаточно памяти |
| FETCH_DIR | -2 | Неверный параметр "указатель записи" в функциях группы SQLFetch |
| WRONG_COUNT_PARAM | -3 | Неверное число параметров |
| WRONG_BIND_PARAM | -4 | Неверный тип подставляемого параметра для функций SQLExecute или SQLBindParameter |
| WRONG_TYPE_PARAM | -5 | Неверный тип параметра |
| WRONG_NUMBER_PARAM | -6 | Неверный номер параметра |
| NO_BLOB_COLUMN | -7 | Не BLOB-столбец |
| UNKNOWN_OPTION | -8 | Неизвестная опция |

# Указатель методов


**A**
AddBlob, 36

**C**
Close, 11, 13
Create, 7

**D**
DelBlob, 37

**F**
Free, 11

**G**
GetAnswerInfo, 31
GetBindParamInfo, 21
GetBlob, 35
GetDBInfo, 13
GetLastAutoInc, 34
GetLastRowId, 33
GetProcOutParams, 25
GetRowCount, 24
GetStatus, 10

**O**
Open, 7, 12

**S**
SetOption, 9
SQLBindParameter, 22
SQLColumns, 16
SQLExecute, 23
SQLExecuteDirect, 19
SQLFetchHash, 27
SQLFetchManyHash, 30
SQLFetchManyRow, 29
SQLFetchRow, 26
SQLNativeSql, 18
SQLPrepare, 20
SQLTables, 14


 © Ruby-интерфейс. АО НПП «РЕЛЭКС», 1990-2026