СИСТЕМА
УПРАВЛЕНИЯ
БАЗАМИ
ДАННЫХ

ЛИНТЕР®

ЛИНТЕР БАСТИОН

ЛИНТЕР СТАНДАРТ

Python-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

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 прикладного (API) интерфейса между СУБД ЛИНТЕР и Python-программой.

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 для работы с СУБД ЛИНТЕР на языке программирования Python.

Необходимые предварительные знания

Для работы необходимо:

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

Дополнительные документы

- СУБД ЛИНТЕР. Справочник по SQL
- СУБД ЛИНТЕР. Справочник кодов завершения

Общие сведения

Назначение

Прикладной интерфейс (API) между СУБД ЛИНТЕР и Python-программой обеспечивается Python-драйвером, который представляет собой динамически подгружаемую библиотеку, написанную полностью на языке программирования Си.

Драйвер основывается на спецификации Python Database API Specification v2.0 (https://www.python.org/dev/peps/pep-0249/).

Драйвер экспортирует описанные ниже функции и свойства, доступные из Python-программы.

Перед установкой Python-драйвера должен быть установлен `Python` и установлено значение переменой среды окружения `PythonPATH`.

В дистрибутивах СУБД ЛИНТЕР для ОС Windows поставляются собранные динамические библиотеки для разных версий Python (от 2.2 до 3.4 – Linter\intlib\Python\) – `LinPy.pyd`, для ОС UNIX имя файла – `LinPy.so`.

Для загрузки драйвера в Python-программу необходимо вставить следующую строку:

```
import LinPy
```

(при этом файл `LinPy.so` должен находиться в пути переменной среды окружения `PythonPATH` или в стандартном пути `Python`).

Для успешной загрузки LinPy в ОС типа Windows необходимо, чтобы путь до `Linter\bin` был прописан в переменной окружения PATH.

Python-драйвер СУБД ЛИНТЕР функционирует в среде ОС Windows и в среде ОС UNIX.

Условия применения

Перед установкой Python-драйвера должен быть установлен Python и установлено значение переменой среды окружения PythonPATH.

Для корректной работы драйвера необходимо указать в переменной окружения PATH местоположение используемых драйвером библиотек СУБД ЛИНТЕР `inter325.dll` (`inter64.dll`), `lapi325.dll` (`lapi64.dll`) и `dectic32.dll` (`dectic64.dll`). Библиотеки располагаются в подкаталоге /bin установочного каталога СУБД ЛИНТЕР.

Если в системе определена переменная окружения LINTER_HOME, достаточно добавить в переменную окружения PATH путь %LINTER_HOME%\bin.

Сборка и установка драйвера

Python-интерфейс СУБД ЛИНТЕР поддерживает версии 3.x.x.

Версии драйвера Python 2.x.x не поддерживаются.

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

Сборка и установка для ОС типа Windows

Запустить консоль Visual Studio, перейти в подкаталог /intlib/python установочного каталога СУБД ЛИНТЕР.

Для сборки драйвера выполнить команду:

```
python setup.py build
```

Для установки собранного драйвера выполнить команду:

```
python setup.py install
```

Примечание
Если при выполнении команды python setup.py build было выдано сообщение «`Unable to find vcvarsall.bat`», то необходимо:

1. проверить, установлен ли в Visual Studio компонент Visual C++;
2. выполнить одну из команд в соответствии с используемой VS:

```
Visual Studio 2010 (VS10): SET VS90COMNTOOLS=%VS100COMNTOOLS%
Visual Studio 2012 (VS11): SET VS90COMNTOOLS=%VS110COMNTOOLS%
Visual Studio 2013 (VS12): SET VS90COMNTOOLS=%VS120COMNTOOLS%
Visual Studio 2015 (VS14): SET VS90COMNTOOLS=%VS140COMNTOOLS%
Visual Studio 2017 (VS15): SET VS90COMNTOOLS=%VS150COMNTOOLS%
```

В результате библиотека `LinPy.pyd` будет собрана в python_path\Lib\site-packages.

Сборка и установка для ОС типа UNIX

Для сборки Python-драйвера СУБД ЛИНТЕР:

1) запустить программу (скрипт) конфигурации configure из корневого каталога дистрибутива СУБД ЛИНТЕР;
2) ответить утвердительно на вопрос о конфигурации дистрибутива для сборки Python-драйвера;
3) определить местоположение заголовочных файлов Python:
   - методом поиска файлов;
   - явным указанием полного пути (если он известен до выполнения данной операции). Если при вводе была допущена ошибка (по указанному пути файлы не найдены), будет предложено повторить ввод. Отказ от повторного ввода равносилен отказу от сборки Python-интерфейса;
   - отредактировать файл `Definition` дистрибутива СУБД ЛИНТЕР вручную, установив в переменной окружения Python_INC полный путь к заголовочным файлам Python.
4) перейти в подкаталог /python установочного каталога СУБД ЛИНТЕР и выполнить команду:

```
make
```

5) для сборки драйвера перейти в подкаталог /python установочного каталога СУБД ЛИНТЕР и выполнить команду:

```
python setup.py build
```

Для установки собранного драйвера выполнить команду:

```
python setup.py install
```

В результате библиотека LinPy будет помещена в необходимый подкаталог установочного каталога СУБД ЛИНТЕР.

Функции интерфейса

Соединение с БД

Назначение

Создание объекта типа Connection и соединение с БД.

Синтаксические правила

LinPy.connect(user = <пользователь>,password =<пароль> [,database = <сервер>][,mode = <режим>])

<пользователь> – имя пользователя БД

<пароль> – пароль пользователя БД

<сервер> – имя ЛИНТЕР-сервера (узла локальной сети, на котором находится БД). Если аргумент не задан или имеет пустое значение, то соединение осуществляется с локальной БД

<режим> – уровень изоляции транзакций по данному соединению:

- LinPy.M_EXCLUSIVE;
- LinPy.M_OPTIMISTIC;

Примечание
Режим M_OPTIMISTIC устарел (использовать не рекомендуется).

- LinPy.M_AUTOCOMMIT.

Возвращаемое значение

Идентификатор соединения с БД (в случае успешного соединения).

Примечания

1. В случае ошибки возникнет исключение, функция не завершится и код возврата не сформирует. Это поведение относится ко всем функциям Python-интерфейса.
2. По умолчанию установлен режим M_EXCLUSIVE.

Примеры

```
# подсоединение пользователя SYSTEM с паролем MANAGER8 к локальной
 БД
connection = LinPy.connect('SYSTEM', 'MANAGER8')

# подсоединение пользователя BORIS с паролем 123 к БД UNCLE
# в режиме автофиксации изменений
connection = LinPy.connect(
user = 'BORIS',
password = '123',
```

```
database = 'UNCLE',
mode = LinPy.M_AUTOCOMMIT)
```

Глобальные переменные

Python-программе доступны следующие предопределенные глобальные переменные:

`apilevel`

Строковая константа, содержащая поддерживаемый уровень DB API. В настоящее время содержит строку '2.0'.

`threadsafety`

Целочисленная константа, обозначающая уровень нитебезопасности, поддерживаемый данным интерфейсом. Допустимое значение `1`, т. е. нити могут разделять модуль, но не соединения.

`paramstyle`

Строковая константа, задающая тип представления параметров, ожидаемый интерфейсом, в параметрических SQL-запросах. Допустимое значение 'qmark', т. е. параметр может быть либо позиционным (стиль '...WHERE "Имя"=?'.), либо именованным (стиль '...WHERE "Имя"=:name').

Примечание
Предпочтительным (по соображениям быстродействия) является использование именованных параметров.

`version`

Строковая константа, обозначающая версию Python-драйвера.

Исключения

В процессе выполнения Python-программа выдает информационные и диагностические сообщения с помощью следующих исключений:

`Warning`

Исключение, вызываемое для важных предупреждений, таких, как усечение данных в процессе вставки и т.д. Этот класс является производным от класса `Python StandardError` (определенного в модуле `exceptions`).

`Error`

Исключение, являющееся базовым классом всех других исключений, связанных с ошибками. Его можно использовать для перехвата всех исключений с помощью единственного утверждения `'except'`. Предупреждения не рассматриваются как ошибки и поэтому не должны использовать этот класс в качестве базового. Этот класс является производным от класса `Python StandardError` (определенного в модуле exceptions).

`InterfaceError`

Исключение порождается ошибками в Python-интерфейсе. Является производным от класса `Error`.

`DatabaseError`

Исключение порождается ошибками обработки данных в БД. Оно является подклассом класса `Error`.

`DataError`

Исключение порождается при обработке некорректных данных в БД (деление на ноль, выход величины за пределы допустимых значений и т.д.). Является производным от класса `DatabaseError`.

`OperationalError`

Порождение исключения вызывается внешними причинами (неожиданный обрыв соединения, не найдено имя источника данных, невозможно обработать транзакцию, ошибка распределения памяти в процессе обработки и т.д.). Является производным от класса `DatabaseError`.

`IntegrityError`

Исключение порождается при нарушении ссылочной целостности БД (например, попытка ссылки на несуществующую внешнюю запись). Является производным от класса `DatabaseError`.

`InternalError`

Исключение порождается внутренними ошибками СУБД (например, курсор больше не является допустимым, транзакция не синхронизирована и т.д.). Является производным от класса `DatabaseError`.

`ProgrammingError`

Исключение порождается программными ошибками (например, таблица не найдена или уже существует, синтаксическая ошибка в SQL-предложении, задано неверное число параметров и т.д.). Является производным от класса `DatabaseError`.

`NotSupportedError`

Исключение порождается в случае использования вызова API. Является производным от класса `DatabaseError`. Схема наследования исключений показана на рисунке.

```
StandardError
|__Warning
|__Error
    |__InterfaceError
    |__DatabaseError
         |__DataError
         |__OperationalError
         |__IntegrityError
         |__InternalError
         |__ProgrammingError
         |__NotSupportedError
```

Рисунок. Схема наследования исключений

Примечание
Все исключения, возникающие в Python-драйвере, содержат коды завершения и их текстовую расшифровку. Для получения дополнительных сведений об исключении можно

воспользоваться следующими расширенными свойствами, не входящими в спецификацию DB API 2.0:

`-code`	код завершения СУБД ЛИНТЕР
`-message`	текстовая расшифровка кода завершения
`-linCode`	синоним свойства `code`
`-apiCode`	код ошибки прикладного интерфейса
`-sysCode`	код системной ошибки
`-object`	объект, вызвавший исключение

Пример

```
try:
  con = LinPy.connect('SYSTEM', 'MANAGER8', '', LinPy.M_EXCLUSIVE)
  cur = LinPy.cursor(con)
…

except LinPy.DatabaseError as e:
  error, = e.args
  print(error.code)
  print(error.apiCode)
  print(error.linCode)
  print(error.sysCode)
  print(error.message)
```

Атрибуты и методы класса Connection

Атрибуты класса

Доступные атрибуты класса приведены в таблице 1 .

Таблица 1. Доступные атрибуты соединения

Имя атрибута	Тип доступа	Значение атрибута
username	read-only	Имя пользователя, который установил соединение
database	read-only	Имя ЛИНТЕР-сервера, с которым было установлено соединение
mode	read/write	Уровень изоляции транзакции
version	read-only	Версия СУБД ЛИНТЕР

Методы класса

Создание курсора

Назначение

Создание нового объекта курсора для соединения.

Синтаксические правила

Вариант 1:

```
LinPy.cursor(connection[, <режим>]|[mode = <режим>])
```

Вариант 2:

```
connection.cursor([<режим>]|[mode = <режим>])
```

<режим> – уровень изоляции транзакций для курсора:

- LinPy.M_EXCLUSIVE;
- LinPy.M_OPTIMISTIC;

Примечание
Режим M_OPTIMISTIC устарел (использовать не рекомендуется).

- LinPy.M_AUTOCOMMIT.

Если <режим> не задан, по умолчанию используется ранее установленный режим соединения.

Возвращаемое значение

Объект типа cursor – в случае успешного создания.

None – в случае неуспешного завершения метода.

Подробнее см. подраздел «Атрибуты и методы класса Cursor».

Примеры

```
1.
connection= LinPy.connect('SYSTEM', 'MANAGER8')
cursor = LinPy.cursor(connection, mode = LinPy.M_EXCLUSIVE)
2.
connection= LinPy.connect('SYSTEM', 'MANAGER8')
cursor = connection.cursor()
```

Подтверждение транзакции

Назначение

Подтверждение изменений, внесенных в БД по соединению и подчиненным курсорам.

Синтаксические правила

```
connection.commit()
```

Описание

По умолчанию установлен режим M_EXCLUSIVE. Для автофиксации изменений в БД необходимо указать режим M_AUTOCOMMIT в методе connect().

Возвращаемое значение

Нет.

Отмена транзакции

Назначение

Откат изменений, внесенных в БД по соединению и подчиненным курсорам.

Синтаксические правила

```
connection.rollback()
```

Описание

Метод заставляет СУБД выполнить откат к началу всех незавершенных транзакций по этому соединению. Закрытие соединения без предварительной явной фиксации изменений приведет к неявному откату.

Возвращаемое значение

Нет.

Закрытие соединения

Назначение

Закрытие соединения.

Синтаксические правила

```
connection.close()
```

Описание

Метод выполняет немедленное закрытие соединения (а не отложенное, как при вызове метода `__del__()`). Соединение становится недоступным для последующего использования. При попытке выполнить по нему некоторую операцию (в том числе курсорную) порождается соответствующее исключение.

Возвращаемое значение

Нет.

Атрибуты и методы класса Cursor

Объекты класса Cursor представляют курсор СУБД, используемый для управления контекстом операции выборки.

Атрибуты класса

Доступные атрибуты класса приведены в таблице 2.

Таблица 2. Атрибуты курсора

Имя атрибута	Тип доступа	Значение атрибута
description	read-only	Описание столбцов запроса выборки данных в виде массива: (<описание столбца1>) (<описание столбца2>)… Элемент массива имеет следующую структуру: • name – имя столбца; • type_code – тип данных столбца (анализируется путем сравнения с объектами типа); • display_size – длина отображаемого значения (для строковых значений); • display_size – длина отображаемого значения (для строковых значений); • precision – точность (для вещественных значений); • scale – масштаб (для вещественных значений); • null_ok – допустимость NULL-значений (1 – да, 0 – нет). Пример: `[('CHARS_COL', <type 'STRING'>, 20, 20, 0, 0, 1), ('VARCHARS_COL', <type 'STRING'>, 20, 20, 0, 0, 1)]`
rowcount	read-only	Количество реально обработанных строк: • выбранных (в SQL-запросах типа SELECT); • модифицированных (в SQL-запросах типа UPDATE, INSERT) последним вызовом executeXXX(); • удаленных (в SQL-запросах типа DELETE). Атрибут принимает значение `-1` в случае, если над курсором не выполнен ни один вызов executeXXX() или если Python-интерфейс не может определить количество обработанных строк в последнем выполненном SQL-запросе.
arraysize	read/write	Количество строк, выбираемых с помощью метода fetchmany() за один вызов. Если не установлено, по умолчанию равно `1`
mode	read-only	Уровень изоляции курсорных транзакций
name	read/write	Имя курсора для операций с конструкцией WHERE CURRENT OF
statement	read-only	Текст SQL-запроса, оттранслированного с помощью метода prepare() или выполненного с помощью метода execute()
connection	read-only	Ссылка на соединение, от которого создан курсор
channelid	read-only	Идентификатор соединения

Примечание

Атрибут объекта курсора description возвращает информацию о каждом из столбцов результата запроса. Код типа type_code должен быть проверен на равенство одному из типов python-объектов. Объекты типа могут быть равны более чем одному коду типа

(например, DATETIME может быть равен кодам типа для столбцов даты, времени и временной метки).

Методы класса

Выполнение запроса

Назначение

Подготовка и выполнение SQL-запроса.

Синтаксические правила

```
cursor.execute(<SQL-запрос>[,<параметры>])

<SQL-запрос> ::= символьная строка

<параметры> ::= <окс><python-элемент типа tuple> <зкс>
                  | <опс> <python-элемент типа list> <зпс>
                  | <офс> <python-элемент типа dict> <зфс>
<окс>::= (
<зкс>::= )
<опс>::= [
<зпс>::= [
<офс>::= {
<зфс>::= }
<python-элемент типа tuple> ::=
см. спецификацию языка программирования Python
<python-элемент типа list> ::=
см. спецификацию языка программирования Python
<python-элемент типа dict> ::=
см. спецификацию языка программирования Python
```

Примечание
Синтаксис SQL-запросов (в том числе параметризованных) описан в документе «СУБД ЛИНТЕР. Справочник по SQL» (см. также атрибут paramstyle класса Connection).

Описание

Ссылка на SQL-запрос сохраняется курсором. Если объект операции SQL-запроса передается снова на обработку, курсор может оптимизировать свое поведение (не выполнять повторную трансляцию SQL-запроса, а использовать его претранслированный вариант). Это наиболее эффективно для алгоритмов, в которых много раз используется одна и та же операция с привязкой различных параметров.

Привязка значений к неименованным параметрам выполняется в порядке следования значений:

1) в случае передачи параметров в виде python-элемента типа tuple:

```
cursor.execute("insert into PEOPLE (ID, NAME) values (?, ?)", (1,
 "BORIS"))
```

2) в случае передачи параметров в виде python-элемента типа list:

```
cursor.execute("insert into PEOPLE (ID, NAME) values (?, ?)", [1,
 "BORIS"])
```

В результате в обоих вариантах будет добавлена строка со значением полей:

```
ID=1, NAME="BORIS", BDAY= значение по умолчанию (если определено
 или NULL).
```

3) привязка значений к именованным параметрам выполняется с помощью python-элемента типа dict, например:

```
cursor.execute("insert into PEOPLE (ID, NAME, BDAY) values
 (:I, :N, :D)", {"I" : 1, "N" : "BORIS", "D" : LinPy.Date(1980, 5,
 27)})
```

В результате будет добавлена строка со значением полей:

```
ID=1, NAME="BORIS", BDAY= LinPy.Date(1980, 5, 27)
```

Возвращаемое значение

Нет.

Пакетная обработка данных

Назначение

Подготовка и выполнение SQL-запроса в режиме пакетной обработки.

Синтаксические правила

cursor.executemany (<SQL-запрос>[,<параметры> …])

```
<SQL-запрос> – символьная строка
<параметры>  ::= <опс>
                   <python-элементы типа tuple> |
                   <python-элементы типа list>  |
                   <python-элементы типа dict>
                   <зпс>
<python-элементы типа tuple>::= <окс> tup1,..,tupN <зкс>
<python-элементы типа list>::= <опс> list1,..,listN <зпс>
<python-элементы типа dict>::= <офс> dict1,..,dictN <офс>
N – количество пакетов к обработке.
<окс>::= (
<зкс>::= )
<опс>::= [
<зпс>::= [
<офс>::= {
```

```
<зфс>::= }
<tupN> – python-элемент типа tuple (см. спецификацию языка
 программирования Python)
<listN> – python-элемент типа list (см. спецификацию языка
 программирования Python)
<dictN> – python-элемент типа dict (см. спецификацию языка
 программирования Python)
```

Примечание
Синтаксис SQL-запросов (в том числе параметризованных) описан в документе «СУБД ЛИНТЕР. Справочник по SQL» (см. также атрибут paramstyle класса Connection).

Описание

Метод позволяет за один вызов подготовить SQL-запрос и затем выполнить его для всего пакета данных.

Пакет данных представляет собой массив, каждый элемент которого содержит набор значений параметров исполняемого SQL-запроса.

Значения параметров должны быть представлены в одном стиле (смешение разных типов представления не допускается).

Возвращаемое значение

Нет.

Примеры

```
1) передача параметров в виде python-элементов типа tuple:
cursor.executemany("insert into PEOPLE (ID, NAME) values (?, ?)",
[(1, "BORIS"),
(2, "SASHA")])
```

```
2) передача параметров в виде python-элементов типа list:
cursor.executemany("insert into PEOPLE (ID, NAME) values (?, ?)",
[[1, "BORIS"],
[2, "SASHA"]])
```

```
3) передача именованных параметров с помощью python-элементов типа
 dict:
cursor.executemany("insert into PEOPLE (ID, NAME) values
 (:I, :N)",
[{"I" : 1, "N" : "BORIS"},
{"I" : 2, "N" : "SASHA"}])
```

Получение всей выборки данных

Назначение

Получение всех записей запроса выборки данных.

Синтаксические правила

cursor.fetchall()

Описание

Метод предоставляет выборку данных целиком.

Атрибут курсора `arraysize` влияет на производительность этой операции.

Исключение `Error` (или его подкласс) порождается, если запрос выборки ранее не подавался.

Возвращаемое значение

1) Выборка данных.
2) `None` (выборка данных пуста).

Пример

```
cursor.execute('select make from auto fetch first 5')
print(cursor.fetchall())
|ALPINE    |
|AMERICAN MOTORS  |
|MASERATI|
|CHRYSLER|
|MERCEDES-BENZ|
```

Получение следующей записи выборки данных

Назначение

Получение следующей записи запроса выборки данных.

Синтаксические правила

cursor.fetchone()

Описание

Метод предоставляет следующую запись текущей выборки данных.

Исключение `Error` (или его подкласс) порождается, если запрос выборки ранее не подавался.

Возвращаемое значение

1) Следующая запись запроса выборки данных. Если после выполнения курсорного запроса строки не выбирались, выдается первая строка выборки данных.
2) `None` (строк в выборке данных больше нет).

Пример

```
cursor.execute('select distinct make from auto')
```

```
print(cursor.fetchone())
| ALPINE    |
print(cursor.fetchone())
|AMERICAN MOTORS  |
print(cursor.fetchone())
| BMW           |
```

Получение следующей порции записей выборки данных

Назначение

Получение следующей порции записей выборки данных.

Синтаксические правила

cursor.fetchmany([<количество>])

<количество> – количество записей для выборки за один вызов. Если аргумент не задан, число записей для выборки определяется атрибутом курсора arraysize.

Описание

Метод предоставляет следующую порцию записей запроса выборки данных.

Если количество запрошенных записей превышает количество выбранных (при первом вызове метода) или оставшихся (после предыдущих вызовов метода), может быть возвращено меньшее число записей.

Исключение Error (или его подкласс) порождается, если запрос выборки ранее не подавался.

Для достижения оптимальной производительности в большинстве случаев лучше не указывать размер порции, а пользоваться атрибутом курсора arraysize. Если размер порции указывается, то желательно, чтобы он сохранял одну и ту же величину при всех вызовах метода fetchmany().

Возвращаемое значение

1) Заданное или имеющее количество записей.
2) Пустая последовательность, если выборка данных исчерпана.

Пример

```
cursor.arraysize = 3
cursor.execute ('select distinct make from auto');
print(cursor.fetchmany(3))
|ALPINE              |
|AMERICAN MOTORS     |
|BMW                 |

print(cursor.fetchmany(2))
|CHRYSLER            |
|CITROEN             |
```


Переход к следующему подмножеству выборки данных

Назначение

Перемещение курсора к следующему подмножеству пакетного запроса выборки.

Синтаксические правила

```
cursor.nextset()
```

Описание

Метод заставляет курсор перейти на следующее имеющееся подмножество пакетного запроса выборки, игнорируя все оставшиеся записи текущего подмножества.

Исключение порождается, если выборка данных ранее не выполнялась.

Примечание
В данной версии python-интерфейса метод не реализован, т.к. СУБД ЛИНТЕР не поддерживает возможность генерации нескольких выборок по одному курсору.

Возвращаемое значение

1) `TRUE` – курсор перемещен к следующему подмножеству.
2) `None` – список подмножеств выборки исчерпан.

Если метод возвращает `TRUE`, то последующие обращения к методам выборки будут возвращать записи из следующего подмножества выборки.

Блокирование текущей строки курсора

Назначение

Блокирование строки в текущей позиции курсора.

Синтаксические правила

```
cursor.lockrow()
```

Описание

Метод осуществляет блокировку текущей записи таблицы (или обновляемого представления). Текущая запись определяется как последняя выбранная строка в курсоре.

Возвращаемое значение

Нет.

Разблокирование текущей строки курсора

Назначение

Разблокирование строки в текущей позиции курсора.

Синтаксические правила

cursor.unlockrow()

Описание

Метод осуществляет разблокирование ранее заблокированной текущей записи таблицы. Текущая запись определяется как последняя выбранная строка в курсоре.

Возвращаемое значение

Нет.

Трансляция запроса

Назначение

Трансляция SQL-запроса.

Синтаксические правила

cursor.prepare(<SQL-запрос>)

<SQL-запрос> – символьная строка.

Примечание
Синтаксис SQL-запросов (в том числе параметризованных) описан в документе «СУБД ЛИНТЕР. Справочник по SQL» (см. также атрибут paramstyle класса Connection).

Привязка параметров к претранслированному запросу

Назначение

Привязка параметров к претранслированному ранее SQL-запросу.

Синтаксические правила

cursor.setvalue(<номер>, <спецификация значений>)

<номер> – номер параметра (нумерация начинается с нуля).

<спецификация значений> ::= (<элемент1>,..<элементN>).

<элемент1>,..<элементN> – значения переменных соответствующего <номера>.

N – заданное число выполнений запроса.

Описание

Выполняется привязка переданных значений к последнему претранслированному по соединению SQL-запросу.

Для столбцов запроса, допускающих null-значения, привязку параметра можно не делать (при выполнении запроса будут использованы null-значения).

Возвращаемое значение

Нет.

Выполнение претранслированного запроса

Назначение

Выполнение претранслированного ранее SQL-запроса.

Синтаксические правила

cursor.executemanyprepared(<количество выполнений>)

<количество выполнений> – целочисленное положительное значение.

Описание

<Количество выполнений> задает количество выполнений претранслированного запроса с разными значениями параметров. Для INSERT SQL-запросов допустимо использование меньшего значения <количество выполнений>, чем указано количество значений для каждого параметра в методе setvalue.

Выполняется претранслированный SQL-запрос с ранее предварительно привязанными параметрами.

Возвращаемое значение

Нет.

Пример

```
cursor.prepare("insert into PEOPLE (ID, NAME) values (?, ?)")
cursor.setvalue(0, (0, 1))
cursor.setvalue(1, ('Alex', 'Sergey'))
cursor.executemanyprepared(2)
```

Выполнение хранимой процедуры

Назначение

Вызов на выполнение хранимой процедуры БД.

Синтаксические правила

cursor.callproc([<имя схемы>.]<имя процедуры>[,<список параметров>])

Описание

<Имя схемы> – схема, в которой создана процедура. Если <имя схемы> совпадает с именем пользователя, который вызывает процедуру, то указывать <имя схемы> не обязательно.

<Список параметров> – python-список типа tuple, содержащий значение для каждого параметра.

Возвращаемое значение

Возвращается модифицированная копия переданных параметров. Input-параметры остаются нетронутыми, Output и input/output параметры могут быть заменены новыми значениями. Процедура может вернуть записи выборки данных, доступ к которым возможен через стандартные методы fetchXXX().

Пример

```
result = cursor.callproc("myproc", ("hi", 5, var))
```

Выделение памяти для параметров запроса

Назначение

Предварительное выделение памяти для хранения значений параметров SQL-запросов.

Синтаксические правила

cursor.setinputsizes(<размеры>)

<размеры>::=<тип параметра> | <значение>

<тип параметра> – спецификация типа параметра, для которого запрашивается выделение памяти (например, IntType);

<значение> – максимальная длина параметра строкового типа (в символах).

Описание

Если в спецификации размера параметра указано none, предварительное выделение памяти под значение параметра выполняться не будет (это бывает полезным при работе с большими объемами данных).

Метод может использоваться перед вызовом метода execute().

Возвращаемое значение

Нет.

Примечание
В данной версии python-интерфейса метод не поддерживается.

Выделение памяти для BLOB-значений

Назначение

Определение размера буфера для загрузки значений BLOB-столбцов.

Синтаксические правила

cursor.setoutputsize(<размер>[,<столбец>)

<размер> – размер буфера для хранения значений BLOB-столбца (в байтах).

<столбец> – номер BLOB-столбца, для которого выделяется память.

Описание

Номер столбца задается как индекс в запросе выборки. Если столбец не задан, устанавливается <размер> по умолчанию для всех BLOB-столбцов в курсоре.

Метод может использоваться перед вызовом метода executeXXX().

Примечание
Функция может использоваться как до вызова функции execute(), так и после. Если для BLOB-столбца установлен возвращаемый размер порции данных, то функции fetchXXX() будут возвращать непосредственно BLOB-данные вместо объекта BLOB. Чтобы отменить заданное значение возвращаемой порции данных для BLOB-столбцов, необходимо в качестве значения размера передать None, например:

1. setoutputsize(None) – отменяет заданный размер для всех BLOB-столбцов, теперь функции fetchXXX() будут возвращать объекты BLOB;
2. setoutputsize(None, 2) – отменяет заданный размер для второго BLOB-столбца в выборке, теперь функции fetchXXX() будут возвращать для этого столбца объект BLOB.

Возвращаемое значение

Нет.

Подтверждение курсорной транзакции

Назначение

Подтверждение транзакции по курсору.

Синтаксические правила

```
cursor.commit()
```

Описание

Метод аналогичен методу commit() для соединения, но производит фиксацию изменений в БД только по курсору.

Отмена курсорной транзакции

Назначение

Отмена транзакции по курсору.

Синтаксические правила

```
cursor.rollback()
```

Описание

Метод аналогичен методу rollback() для соединения, но производит откат изменений в БД только по курсору.

Закрытие курсора

Назначение

Закрытие курсора.

Синтаксические правила

cursor.close()

Описание

Метод выполняет немедленное закрытие курсора (а не отложенное, как при вызове метода `__del__()`). С момента вызова и далее курсор становится неиспользуемым; в случае попытки обратиться к некоторой операции курсора порождается соответствующее исключение.

Атрибуты и методы класса BLOB

Атрибуты класса

Для BLOB-столбцов доступны следующие атрибуты для чтения:

1) length – текущая длина BLOB-значения;
2) type – тип BLOB-значения.

Пример

```
cur.execute("select I, B from TEST")
print("description: ", str(cur.description))
result = cur.fetchone()
print("result: ", str(result))
result = cur.fetchone()
print( "result: ", str(result))
(i, blob) = result
print("blob length: ", blob.length)
print("blob type: ", blob.type)
```

Методы класса

Если в запросе выборки присутствуют BLOB-столбцы, то значение курсорного атрибута `arraysize` игнорируется – выборка всегда производится по одной записи.

Добавление BLOB-значения

Назначение

Добавление BLOB-значения.

Синтаксические правила

blob.write(<объект>[,<тип>])

<объект> – объект типа buffer, string, unicode, array.

<тип> – целочисленное значение.

Описание

Добавляемое BLOB-значение помещается в конец столбца, к которому относится объект этого метода.

Аргумент <тип> задает тип добавляемых данных (текст, графика, анимация и т. п.). Тип добавляемых данных не контролируется СУБД ЛИНТЕР.

Возвращаемое значение

Нет.

Пример

```
cur.execute("select I, B from TEST where I = 10")
(i, blob) = cur.fetchone()
blob.clear()
blob.write('Hello World')
```

Чтение BLOB-значения

Назначение

Чтение BLOB-значения.

Синтаксические правила

blob.read([<смещение>[,<длина>]])

<смещение> – целочисленное положительное значение.

<длина> – целочисленное положительное значение.

Описание

Аргумент <смещение> задает смещение считываемой порции BLOB-данных (отсчет начинается с 1); если не задан – выборка начинается с начала BLOB-данных.

Аргумент <длина> задает размер (в байтах) считываемой порции BLOB-данных. Если не задан, чтение производится до конца BLOB-значения.

Если заданная длина порции больше длины всего BLOB-значения, то выдается BLOB-значение с заданного смещения и до конца BLOB-значения.

Возвращаемое значение

1) Запрошенная порция BLOB-данных.
2) None – в случае неуспешного завершения метода.

Пример

```
cur.execute("select I, B from TEST")
```

```
print("description: ", str(cur.description))
result = cur.fetchone()
print("result: ", str(result))
result = cur.fetchone()
print("result: ", str(result))
(i, blob) = result
print("blob length: ", blob.length)
print("blob type: ", blob.type)
buf = blob.read(1, blob.length)
print("blob: ",buf)
```

Очистка BLOB-значения

Назначение

Очистка BLOB-значения.

Синтаксические правила

blob.clear()

Возвращаемое значение

Нет.

Пример

```
cur.execute("select I, B from TEST where I = 10")
(i, blob) = cur.fetchone()
blob.clear()
```

Дополнительные типы данных

Создание данных типа «дата»

Назначение

Создание объекта, содержащего значение «дата».

Синтаксические правила

LinPy.Date(<год>,<месяц>,<день>)

<год> – целочисленное положительное значение в диапазоне от 1 до 9999.

<месяц> – целочисленное положительное значение в диапазоне от 1 до 12.

<день> – целочисленное положительное значение в диапазоне от 1 до 31.

Возвращаемое значение

1) Объект типа «дата» (в случае успешного создания).

2) None – в случае неуспешного завершения метода.

Пример

```
# первое января 2003 года
d = LinPy.Date(2003, 1, 1)
cur.execute("INSERT INTO Test(DATE_COL) VALUES(:D)", {"D":d})
```

Создание данных типа «время»

Назначение

Создание объекта, содержащего значение «время».

Синтаксические правила

LinPy.Time(<час>, <минута>, <секунда>[,<тик>])

<час> – целочисленное положительное значение в диапазоне от 0 до 23.

<минута> – целочисленное положительное значение в диапазоне от 1 до 60.

<секунда> – целочисленное положительное значение в диапазоне от 1 до 60.

<тик> – целочисленное положительное значение в диапазоне от 1 до 100.

Возвращаемое значение

1) Объект типа «время» (в случае успешного создания).

2) None – в случае неуспешного завершения метода.

Пример

```
# полдень
t = LinPy.Time(12, 0, 0)
cur.execute("INSERT INTO Test(DATE_COL) VALUES(:D)", {"D":t})
```

Создание данных типа «временна̀я метка»

Назначение

Создание объекта, содержащего значение «временная метка» (timestamp).

Синтаксические правила

LinPy.Timestamp(<год>,<месяц>,<день>,<час>,<минута>,<секунда>[,<тик>])

<год> – целочисленное положительное значение в диапазоне от 1 до 9999.

<месяц> – целочисленное положительное значение в диапазоне от 1 до 12.

<день> – целочисленное положительное значение в диапазоне от 1 до 31.

<час> – целочисленное положительное значение в диапазоне от 0 до 23.

<минута> – целочисленное положительное значение в диапазоне от 1 до 60.

<секунда> – целочисленное положительное значение в диапазоне от 1 до 60.

<тик> – целочисленное положительное значение в диапазоне от 1 до 100.

Возвращаемое значение

1) Объект типа «временная метка» (в случае успешного создания).
2) None – в случае неуспешного завершения метода.

Пример

```
# 5-е апреля 2003 года 6 часов 7 минут 8 секунд и 9 сотых секунды
ts = LinPy.Timestamp(2003, 4, 5, 6, 7, 8, 9)
cur.execute("INSERT INTO Test(DATE_COL) VALUES(:D)", {"D":ts})
```

Создание данных типа «дата» из тиков

Назначение

Создание объекта, содержащего значение «дата», из заданного количества тиков.

Синтаксические правила

LinPy.DateFromTicks(<тики>)

<тики> – целочисленное положительное значение, возвращаемое методом time.time().

Примечание
См. также документацию по стандартному модулю time языка программирования Python.

Возвращаемое значение

1) Объект типа «дата» (в случае успешного создания).
2) None – в случае неуспешного завершения метода.

Пример

```
# текущая дата
d = LinPy.DateFromTicks(time.time())
cur.execute("INSERT INTO Test(DATE_COL) VALUES(:D)", {"D":d})
```

Создание данных типа «время» из тиков

Назначение

Создание объекта, содержащего значение «время», из заданного количества тиков.

Синтаксические правила

LinPy.TimeFromTicks(<тики>)

<тики> – целочисленное положительное значение, возвращаемое методом `time.time()`.

Примечание
См. также документацию по стандартному модулю `time` языка программирования Python.

Возвращаемое значение

1) Объект типа «время» (в случае успешного создания).
2) `None` – в случае неуспешного завершения метода.

Пример

```
# текущее время
t = LinPy.TimeFromTicks(time.time())
cur.execute("INSERT INTO Test(DATE_COL) VALUES(:D)", {"D":t})
```

Создание данных типа «временна̀я метка» из тиков

Назначение

Создание объекта, содержащего значение «временная метка», из заданного количества тиков.

Синтаксические правила

`LinPy.TimeStampFromTicks`(<тики>)

<тики> – целочисленное положительное значение, возвращаемое методом `time.time()`.

Примечание
См. также документацию по стандартному модулю `time` языка программирования Python.

Возвращаемое значение

1) Объект типа «временная метка» (в случае успешного создания).
2) `None` – в случае неуспешного завершения метода.

Пример

```
# текущие время и дата
ts = LinPy.TimestampFromTicks(time.time())
cur.execute("INSERT INTO Test(DATE_COL) VALUES(:D)", {"D":ts})
```

Создание данных типа «байтовая строка»

Назначение

Создание объекта, содержащего значение «байтовая строка». Используется СУБД ЛИНТЕР для работы с данными типа BYTE, VARBYTE.

Синтаксические правила

LinPy.Binary(<строка>)

<строка> – символьная строка шестнадцатеричных цифр.

Возвращаемое значение

1) Объект типа BINARY (в случае успешного создания).
2) None – в случае неуспешного завершения метода.

Пример

```
# байтовая строка
bt = LinPy.Binary('0967fca8dd')
cur.execute("INSERT INTO Test(BYTES_COL) VALUES(:BY)", {"BY":bt})
```

Представление UNICODE-данных

Назначение

Установка режима представления возвращаемых СУБД ЛИНТЕР данных типов char, varchar.

Синтаксические правила

LinPy.SetUnicodeData (<режим>)

<режим> ::= 0 | 1

Описание

Если значение <режима> равно 1, данные возвращаются в виде unicode object, если 0 – в виде bytes object. Значение по умолчанию – значение 0. Актуально для версии python младше 3.

Возвращаемое значение

1) Установленное значение.
2) None – в случае неуспешного завершения метода.

Пример

```
LinPy.SetUnicodeData(1)
```

Формат представления данных типа «дата»

Назначение

Установка режима представления возвращаемых СУБД ЛИНТЕР данных типов «дата».

Синтаксические правила

LinPy.SetDateEmulation (<формат>)

<формат> ::= 0 | 1

Описание

Если значение <формата> равно 0, то значение типа «дата» возвращается в полном виде (datetime.datetime), возможно, с нулевыми значениями календарной даты или времени.

Если значение <формата> равно 1, то значение типа «дата» возвращается в виде:

- datetime.date (если временная часть даты равна 0, т. е. есть только календарная дата);
- datetime.time (если календарная часть даты равна 0, т. е. есть только время);
- datetime.datetime (если дата содержит и календарную, и временную часть).

Значение по умолчанию 0.

Возвращаемое значение

1) Установленное значение.
2) None – в случае неуспешного завершения метода.

Пример

```
d = LinPy.Date(1980, 5, 27)
cur.execute("INSERT INTO Test(DATE_COL) VALUES(:D)", {"D":d})
t = LinPy.Time(12, 30, 11)
cur.execute("INSERT INTO Test(DATE_COL) VALUES(:D)", {"D":t})
ts = LinPy.Timestamp(2003, 4, 5, 6, 7, 8, 9)
cur.execute("INSERT INTO Test(DATE_COL) VALUES(:D)",  {"D":ts})
LinPy.SetDateEmulation(1)
cur.execute("select DATE_COL from  Test")
print(cur.fetchall())

[(datetime.date(1980, 5, 27),), (datetime.time(12, 30, 11),),
 (datetime.datetime(2003, 4, 5, 6, 7, 8, 900),)]
```